NPS AUV workbench: collaborative environment for autonomous underwater vehicles (AUV) mission planning and 3D visualization

Lee, Chin Siong

Monterey, California. Naval Postgraduate School

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

NPS AUV WORKBENCH: COLLABORATIVE ENVIRONMENT FOR AUTONOMOUS UNDERWATER VEHICLES (AUV) MISSION PLANNING AND 3D VISUALIZATION

by

Lee, Chin Siong

March 2004

Thesis Advisor:	Donald P. Brutzman
Thesis Co-advisor:	Curtis L. Blais
Thesis Second Readers:	John Hiles, Duane Davis

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY	**2. REPORT DATE** March 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: NPS AUV Workbench: Collaborative Environment for Mission Planning and 3D Visualization	**5. FUNDING NUMBERS**
6. AUTHOR: Lee, Chin Siong	
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME AND ADDRESS National University of Singapore (NUS) and Defence Science Organization (DSO), Singapore, Office of Naval Research.	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE** A

13. ABSTRACT

The absence of common software platforms for Autonomous Underwater Vehicle (AUV) mission planning and analysis is an ongoing impediment to collaborative work between research institutions, their partners, and end users. This thesis details the design and implementation of a distributable application to facilitate AUV mission planning and analysis. Java-based open-source libraries and a component-based framework provide diverse functionalities. The extensible Markup Language (XML) is used for data storage and message exchange, Extensible 3D (X3D) Graphics for visualization and XML Schema-based Binary Compression (XSBC) for data compression. The AUV Workbench provides an intuitive cross-platform-capable tool with extensibility to provide for future enhancements such as agent-based control, asynchronous reporting and communication, loss-free message compression and built-in support for mission data archiving.

This thesis also investigates the Jabber instant messaging protocol, showing its suitability for text and file messaging in a tactical environment. Exemplars show that the XML backbone of this open-source technology can be leveraged to enable both human and agent messaging with improvements over current systems. Integrated Jabber instant messaging support makes the NPS AUV Workbench the first custom application supporting XML Tactical Chat (XTC).

Results demonstrate that the AUV Workbench provides a capable test bed for diverse AUV technologies, assisting in the development of traditional single-vehicle operations and agent-based multiple-vehicle methodologies. The flexible design of the Workbench further encourages integration of new extensions to serve operational needs. Exemplars demonstrate how in-mission and post-mission event monitoring by human operators can be achieved via simple web page, standard clients or custom instant messaging client. Finally, the AUV Workbench's potential as a tool in the development of multiple-AUV tactics and doctrine is discussed.

14. SUBJECT TERMS AUV Workbench, Virtual Environments, Extensible 3D Graphics, X3D, Scalable Vector Graphics, SVG, Extensible Markup Language, XML, Java, DIS-Java-VRML, Extensible Modeling and Simulation Framework (XMSF), Extensible Messaging and Presence Protocol (XMPP), Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE), Distributed Interactive Simulation (DIS)	**15. NUMBER OF PAGES** 219
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

NPS AUV WORKBENCH: COLLABORATIVE ENVIRONMENT FOR AUTONOMOUS UNDERWATER VEHICLES (AUV) MISSION PLANNING AND 3D VISUALIZATION

Lee, Chin Siong
Civilian, Defence Science and Technology Agency
B.S. (Computer Engineering), Nanyang Technological University, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2004

Author:	Lee, Chin Siong
Approved by:	Don Brutzman Thesis Advisor
	Curtis L. Blais Thesis Co-Advisor
	John Hiles Second Reader
	LCDR Duane T. Davis, USN Second Reader
	Peter Denning Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The absence of common software platforms for Autonomous Underwater Vehicle (AUV) mission planning and analysis is an ongoing impediment to collaborative work between research institutions, their partners, and end users. This thesis details the design and implementation of a distributable application to facilitate AUV mission planning and analysis. Java-based open-source libraries and a component-based framework provide diverse functionalities. The extensible Markup Language (XML) is used for data storage and message exchange, Extensible 3D (X3D) Graphics for visualization and XML Schema-based Binary Compression (XSBC) for data compression. The AUV Workbench provides an intuitive cross-platform-capable tool with extensibility to provide for future enhancements such as agent-based control, asynchronous reporting and communication, loss-free message compression and built-in support for mission data archiving.

This thesis also investigates the Jabber instant messaging protocol, showing its suitability for text and file messaging in a tactical environment. Exemplars show that the XML backbone of this open-source technology can be leveraged to enable both human and agent messaging with improvements over current systems. Integrated Jabber instant messaging support makes the NPS AUV Workbench the first custom application supporting XML Tactical Chat (XTC).

Results demonstrate that the AUV Workbench provides a capable testbed for diverse AUV technologies, assisting in the development of traditional single-vehicle operations and agent-based multiple-vehicle methodologies. The flexible design of the Workbench further encourages integration of new extensions to serve operational needs. Exemplars demonstrate how in-mission and post-mission event monitoring by human operators can be achieved via simple web page, standard clients or custom instant messaging client. Finally, the AUV Workbench's potential as a tool in the development of multiple-AUV tactics and doctrine is discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Don Brutzman for his ideas, in-depth knowledge in the area of XML, AUV modeling and simulation and opportunities, Curt Blais for his guidance on agent technology and XML, John Hiles for introducing the concepts of cognitive blending and LCDR Duane Davis for being a great sounding board and his in-depth domain knowledge, and most importantly, my patient and loving wife, Joanne who has been a great source of strength all through this work. Not forgetting our families back home in Singapore.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

The lack of common software tools for Autonomous Underwater Vehicle (AUV) mission planning and analysis is an ongoing impediment to collaborative work between research institutions, their partners, and end users. Current proprietary software solutions have a myopic view on the capability of AUVs. Most place too much emphasis on single and relatively simple AUV operations. A common software development and mission evaluation platform will not only facilitate modeling and simulation of AUVs, but it will aid in the introduction of complex multi-agent systems to try out and answer more challenging questions. Longer-term needs such as the development of AUV concept of operations and collaborative sensing between vehicles can be achieved.

A common and flexible platform will facilitate the transition from simulation to actual operations.

B. OVERVIEW

This thesis details the design and implementation of a common platform to facilitate AUV mission planning, visualization and analysis. The end product is capable of handling the various phases of a mission. An important component is the definition and use of a common AUV mission control script. The control script defines the AUV commands that are similar to the low-level execution commands that are used by the actual AUV hardware.

Using Java-based open-source libraries for functionality, Extensible Markup Language (XML) for data storage [Serin 2003] and exchange, and a component-based framework, the AUV Workbench provides an intuitive cross-platform-capable tool with extensibility to provide for future enhancements such as agent-based control, asynchronous reporting and communication, and loss-free message compression. As a collaboration environment, it is important that communication channels and tools are easily available for developers and users to communicate. Jabber instant messaging is selected as it is based on open-source Extensible Message and Presence Protocol (XMPP) [XMPP 2004].

In addition, this thesis investigates the Jabber instant messaging protocol and discusses its suitability for text and file messaging in a tactical environment. Exemplars show that the XML backbone of this open-source technology can be leveraged to enable both human and agent messaging with improvements over current systems.

C. MOTIVATION

One motivating factor is to support the current research efforts at NPS and with partners such as Singapore Defence Science Organization (DSO). Similar partner relationships are occurring with other AUV laboratories. A componentized framework using open-source software and open-standards technologies is presented to support collaborative development. The ultimate goal of software components is to fuse the use of different pieces of software into one smoothly operating package. The end product facilitates collaboration and continued research development between the two research entities.

A well-designed and well-documented system promotes knowledge sharing and retention. Ease of use and user interface design is important issues that will determine whether it gains user acceptance. Usability should not be considered as an afterthought. Ultimately user acceptance aids the transition of a modeling and simulation (M&S) tool into a system that meets operational needs.

A further long-term motivation is to develop a common platform that can be extended to become a Sensor Workbench and also support agent research and development. Much important work awaits; that is otherwise impossible without such a dedicated tool. In many respects, the NPS AUV Workbench is the culmination of many technical threads carrying to fruition that were first initiated as part of the NPS AUV Underwater Virtual World. [Brutzman 1994]

D. OBJECTIVES

The primary focus of this thesis is on the design and implementation of a common platform for AUV mission planning and analysis through the use of Open-source software and tools. In addition, this thesis addresses the following research questions:

- What are the open-source tools and open-standards technologies available to facilitate development of a collaborative platform for AUV mission planning and visualization?
- What constitutes an AUV XML-based mission control script?
- Can the mission control script be graphically represented?
- How can open-standards technologies be leveraged to design and implement a message exchange system that can support both human and machine communications?
- Can Jabber be used for machine-to-machine communications; e.g., for self-validating agent-to-agent messaging?
- Can binary files be transported via Jabber instant messaging protocol?
- Can Jabber instant messaging together with HTTP, serve as a reliable means for the transfer of textual and binary data?

E. THESIS ORGANIZATION

This first chapter identifies the purpose and motivation behind conducting this research and establishes the goals for the thesis. Chapter II discusses similar research and provides general background information to the concepts and set of tools and technologies employed in this thesis. Chapter III examines the design and implementation of the various modules that make up the AUV Workbench application. Chapter IV discusses the use of Jabber instant messaging protocol for message exchange of textual and files. It provides an exemplar on how event monitoring can be implemented. Chapter V analyzes the software design of AUV agents. Chapter VI gives a summary of the conclusions and recommendations for future work. The future work section lists eleven specific areas where this thesis can be extended. The appendices present information on the programming source code produced and system installation in conjunction with this thesis. All source code and model content are provided online and in Appendix C.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

This chapter briefly discusses the technologies and tools leveraged in the conduct of this thesis. An overview on the open-source tools and open-standards technologies employed is given. This chapter also summarizes pertinent previous work on the current NPS AUV and its virtual world software. Further explanation and study of the topics may be found in the list of references at the end of this thesis.

B. DATA REPRESENTATION AND MANIPULATION USING XML

Data is only as good as the way it is packaged. Information is a valuable asset, but its value depends on its longevity, flexibility, and accessibility. Traditionally, data is represented in a simple text-based format (see Table 1). The main disadvantage of such an approach is that it is likely to introduce ambiguity in how the data is captured, resulting in additional effort to write a robust parser to handle the ambiguity. This parser has to handle case-sensitivity ("WAYPOINT" is not the same as "Waypoint") and potential variations in user input (e.g., "WAYPOINT" and "WAYPOINT-ON" refer to the same information). This added logic slows down (and may confuse) in-water processing time.

```
WAYPOINT    #X #Y [#Z]  [#rpm]
WAYPOINT-ON #X #Y [#Z]  [#rpm]
                  Point towards waypoint with coordinates (#X, #Y)
                  (depth #Z optional) (speed  #rpm optional).  You can
                  leave waypoint control by ordering course, rudder,
                  sliding-mode, rotate or lateral thruster control.

                  If speed is < 200 RPM, port & starboard RPMs are
                  increased to  400 RPM to ensure waypoint can be
                  achieved.

                  If in TACTICAL mode, execution reports STABLE when
                  waypoint is achieved.
```

Table 1. Description of text-based command for Waypoint orders. Extracted from mission.script.HELP [Brutzman 1994].

The World Wide Web Consortium's (W3C's) XML Working Group developed Extensible Markup Language (XML) in 1996. XML evolved out of the earlier Standard Generalized Markup Language (SGML), HyperText Markup Language (HTML), and the earliest presentation markup language. XML documents contain only data, not formatting instructions. XML is an open standard and its extensibility allows it to markup virtually any type of information. XML is a simple, standard way to interchange structured textual data between applications. It is also readable and writable by humans, using a simple text editor.

Some examples of XML languages are Extensible HyperText Markup Language (XHTML), Sensor Markup Language (SensorML) for sensors [SensorML], Defense Advanced Research Projects Agency (DARPA) Agent Markup Language (DAML) for agents [DAML], Geography Markup Language to describe geographic information [GeoML], MathML for mathematics [MathML], and Chemical Markup Language (CML) [CML]. A list of XML-based data representation can be found at http://www.xml-acronym-demystifier.org/xmlad.html (Accessed February 2004). The design goals for XML are shown in Table 2 below.

Point	Goal
1.	XML shall be straightforwardly usable over the Internet.
2.	XML shall support a wide variety of applications.
3.	XML shall be compatible with SGML.
4.	It shall be easy to write programs that process XML documents.
5.	The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6.	XML documents should be humanly legible and reasonably clear.
7.	The XML design should be prepared quickly.
8.	The design of XML shall be formal and concise.
9.	XML documents shall be easy to create.
10.	Terseness in XML markup is of minimal importance.

Table 2. XML Design Goals (after W3C, 2003).

1. Removing Ambiguity Through Namespaces

Namespace is a mechanism by which element and attribute names can be assigned to groups. They provide means for document authors to prevent ambiguity and are most often used when combining different vocabularies in the same document. Namespace identifiers have to be assigned some kind of unique identifiers. They are, by convention, assigned to the Uniform Resource Locator (URL) subset of Uniform Resource Identifiers (URIs), not the more abstract Uniform Resource Names (URNs). However, this is not a requirement, since the XML parser does not actually look up any information located at that URL.

2. Defining the XML Document Structure

An XML document can optionally reference a document that defines the document structure and data type. This document can either be represented as Document Type Definition (DTD) or a schema. DTDs were originally developed for XML's predecessor, SGML. They use a compact syntax and provide document-oriented data typing. XML DTDs are a subset of those available in SGML, and the rules for using XML DTDs provide much of the complexity of XML 1.0.

XML Schema is an XML-based alternative to DTD. The XML Schema language is also referred to as XML Schema Definition (XSD). XSD expresses shared vocabularies and allows machines to carry out rules made by people. It provides a means for defining the structure, content and semantics of XML documents [Schema 2004].

Through the use of a schema or DTD, the XML document can be validated (i.e., checked for conformity) as it is parsed. If the XML document follows the DTD or schema, it is valid. If an XML parser can successfully parse an XML document, it means that the document is syntactically correct (well-formed). Therefore a valid XML document is also well-formed.

DTDs are not XML documents (See Figure 1). This makes them difficult to programmatically manipulate. A DTD describes an XML document's structure but not the format of the individual elements. In 1999, the W3C began to develop XML Schemas in response to the growing need for a more advanced format for describing XML documents. XML Schemas reached recommendation status in May 2001.

```
<!—Command the vehicle to transit to a specified location. -->
<!ELEMENT Waypoint EMPTY>
<!—List of attributes -->
<!ATTLIST Waypoint x CDATA><!—CDATA indicates character data -->
<!ATTLIST Waypoint y CDATA>
```

Figure 1. Sample DTD defining a Waypoint element with two attributes “x” and “y”.

```
<xsd:element name=”Waypoint”>
   <xsd:annotation>
      <xsd:appinfo>Command  the  vehicle  to  transit  to  a  specified
location. Vehicle will not stop when location reached.</xsd:appinfo>
   </xsd:annotation>
   <xsd:complexType>
      <xsd:attribute name=”x” type=”xsd:decimal” use=”required”/>
      <xsd:attribute name=”y” type=”xsd:decimal” use=”required”/>
      <xsd:attribute name=”z” type=”xsd:decimal” use=”required”/>
      <xsd:attribute name=”rpm” type=”xsd:decimal” use=”optional”/>
   </xsd:complexType>
</xsd:element>
```

Figure 2. Sample XSD on Waypoint element.

S/N	Functionality	Document Type Definition	XML Schema
1.	Syntax	Extended Backus Naur form.	XML format.
2.	Namespaces	Not fully supported.	Enables the definition of vocabularies that utilize namespace declarations.
3.	Data Types	Text only. No constraint checking.	Simple or complex with constraint checking; e.g., numbers within a certain range, positive numbers or dates.
4.	Entity Declaration	Yes	Yes
5.	Providing defaults for attributes	Yes	Yes
6.	Support embedded declaration	Yes	No
7.	Parser support	Readily supported by most parsers.	Supported by a few open-source parsers (Castor http://www.castor.org, accessed on February 2004)

Table 3. Comparison of DTD and XSD.

3. Transforming XML Documents

As the name implies, Extensible Stylesheet Language (XSL) is intended to define the formatting and presentation of XML documents for display. The first proposal for XSL was dated 21 August 1997 [XSL 2004].

XSL Transformations (XSLT) is a language designed for transforming XML documents into other XML documents [XSL 2004]. Just as XML was derived from SGML, XSLT has its origins in an SGML-based standard, Document Style Semantics and Specification Language (DSSSL). A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added. A transformation expressed in XSLT is called a stylesheet.

XSLT is designed for use as part of XSL, which is a stylesheet language for XML. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary. XSLT is designed to work independently of XSL. The dominant feature of XSLT is that it is declarative. It produces an output when a particular pattern (based on a set of non-sequential template rules) occurs in the input. This is opposed to a procedural program where the tasks are defined in the order they are supposed to perform. Apache Xalan is a Java-based open-source XSLT processor [Xalan 2004] that is used in this thesis.

The basic relationship between an XML document with XSL and XSD is illustrated in Figure 3.

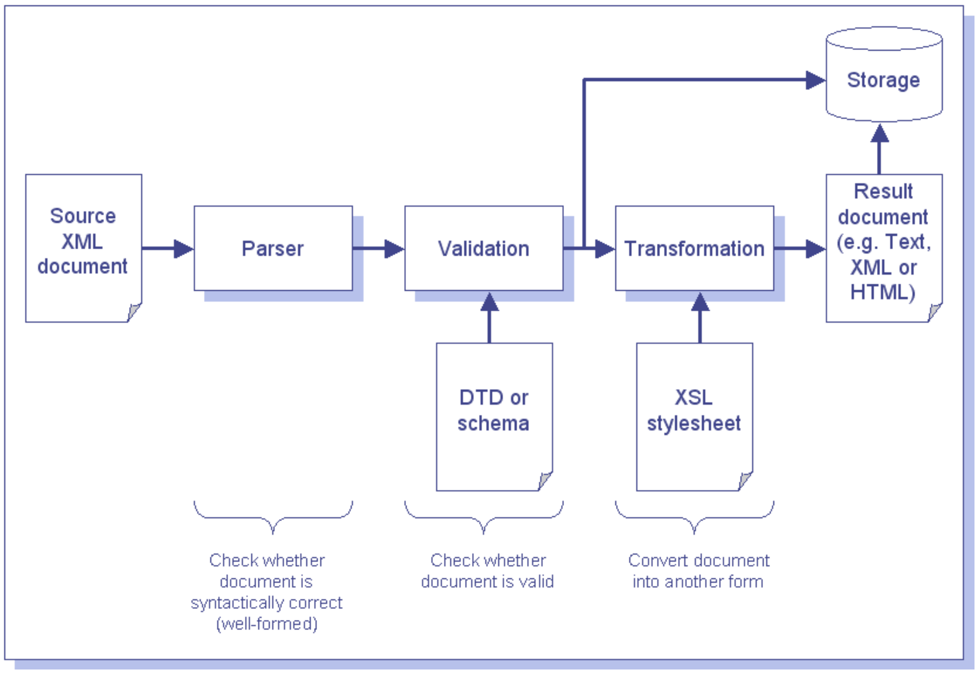

Storage
Source XML document
Parser
Validation
Transformation
Result document (e.g. Text, XML or HTML)
DTD or schema
XSL stylesheet
Check whether document is syntactically correct (well-formed)
Check whether document is valid
Convert document into another form

Figure 3. Relationship of Parsing, Validating and Transforming an XML document.

C. 2D AND 3D GRAPHICS REPRESENTATION

1. Scalable Vector Graphics (SVG)

SVG is a language for describing two-dimensional graphics and graphical applications in XML. It was created by the World Wide Web Consortium (W3C), the non-profit, industry-wide, open-standards consortium that created HTML and XML, among other important standards and vocabularies. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. Text can be in any XML namespace suitable to the application, which enhances searchability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility. As an XML grammar, SVG offers all the advantages of XML. SVG graphics can easily be generated on Web servers "on the fly," using standard XML tools, many of which are written in the Java programming language [SVG 2004].

SVG drawings are dynamic and interactive. The Document Object Model (DOM) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on SVG elements and other XML elements from different namespaces simultaneously within the same Web page.

SVG 1.1 is a W3C Recommendation and forms the core of the current SVG developments. SVG 1.2 is the specification currently being developed.

```
<svg width="360" height="120">
   <rect x="0" y="0" width="100%" height="100%" fill="lightgray"/>
   <g id="sampleLogo" transform="translate(5, 5)">
       <rect fill="#ff3366" width="155" height="70"/>
       <image xlink:href="sample.svg" x="15" y="15"
        width="120" height="40" />
   </g>
   <rect fill="#3366ff" x="165" y="5" width="180" height="70"/>
   <rect fill="#FFFF00" x="10" y="80" width="335" height="35"/>

   <g font-family="SunSansCondensed-Heavy" fill="black"
    font-size="20" stroke="white" >
      <text x="20" y="70" stroke="none" >NPS AUV Workbench</text>
   </g>
</svg>
```

Figure 4. A simple SVG code snippet.

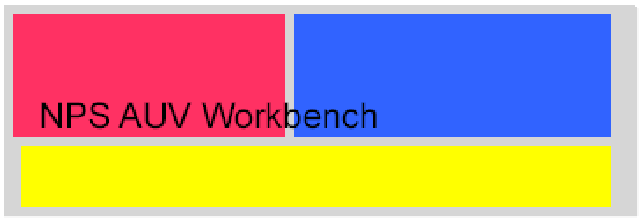


Figure 5. Graphical representation of the above SVG code.

2. Virtual Reality Modeling Language (VRML)

The Virtual Reality Modeling Language (VRML) is am International Standards Organization (ISO) standard for defining 3D virtual worlds through the use of a structured text file, such as depicted in Figure 6. The text files are typically small and are ideal for transmission over the Internet. VRML files typically contain four main types of components; header, prototypes, shapes and routes. VRML virtual worlds are rendered using specialized viewers that read the VRML text files and render the content defined in the file, (e.g., ParallelGraphics Cortona VRML Client 4.2 at http://www.parallelgraphics.com accessed on February 2004). These viewers are installed as Internet browser plug-ins. There are also several open-source VRML viewers available on the Internet, such as Xj3D [XJ3D 2004].

```
#VRML V2.0 utf8
NavigationInfo {
  type [ "EXAMINE" "ANY"  ]
}
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0 0 1
    }
  }
  geometry Box {
    size 1 0.5 1
  }
}
```

Figure 6. Contents of VRML file for a 1m by 0.5m by 1m blue box.

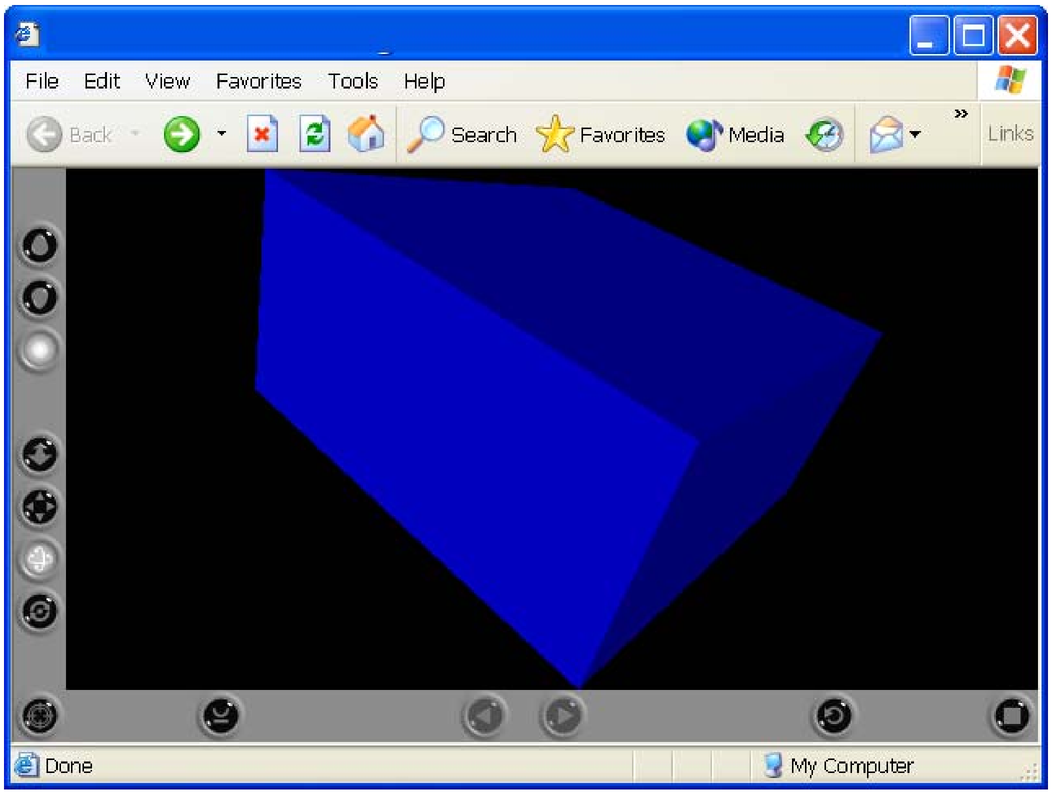


Figure 7. Rendering of the 1m by 0.5m by 1m blue box defined in Figure 6 using Internet Explorer and the Cortona VRML plug-in. User has rotated the scene for a custom viewpoint location and orientation.

3. Extensible 3D (X3D) Graphics

X3D Graphics is the next generation of the Virtual Reality Markup Language 1997 (VRML97) 3D graphics format for the Internet. X3D has been developed with an open-source sample implementation for specification implementation and evaluation along with support from major industry players in 3D content development for the Internet. Since the format is XML based, it can also take advantage of the benefits of XML through the use of XSLT stylesheets to view the same content rendered in VRML97, HTML or with direct rendering of the XML-based tree structure in an open-source browser implementation such as Xj3D [XJ3D 2004].

4. Xj3D 3D Display Library

Xj3D is the open-source rendering implementation for the X3D graphics standard [XJ3D 2004]. It is "a Java-based toolkit developed by Yumetech that allows companies to rapidly support X3D."[X3D 2002] The Web3D Consortium has also formed the Java Rendering Working Group consisting of members from Media Machines Inc. Anaviza Inc., Sun Microsystems, and Yumetech that are concurrently working on the definition and implementation of bindings for various common graphical API's such as OpenGL® and Direct3D™. Upon completion, this implementation will make the specific graphics rendering context of X3D graphics agnostic and this less vulnerable to the commercial "ups and downs" of the market place or consumer popularity.

D. JABBER AND EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL (XMPP)

Jabber is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and other structured information in close to real time. The first Jabber application is an instant messaging (IM) network that offers functionality similar to legacy IM services such as AIM, ICQ, MSN, and Yahoo. However, Jabber is more than just IM, and Jabber technologies offer several key advantages [Jabber 2004]: Jabber protocols are free, open, public, and easily understandable; in addition, multiple implementations exist for clients, servers, components, and code libraries.

First developed by Jeremie Miller in 1998, Jabber is becoming a stable and proven piece of technology. The architecture of the Jabber network is similar to email; as a result, anyone can run their own Jabber server, enabling individuals and organizations to take control of their IM experience. Robust security using Simple Authentication and Security Layer (SASL) and Transport Layer Security (TLS) has been built into the core XMPP specifications.

Using the power of XML namespaces, it is extensible in that anyone can build custom functionality on top of the core protocols; to maintain interoperability, common extensions are managed by the Jabber Software Foundation. Jabber-enabled applications are more than IM. These include network management, content syndication, collaboration tools, file sharing, gaming, and remote systems monitoring.

With a wide range of companies and open-source projects using the Jabber protocols to build and deploy real-time applications and services; there is no technology "locked in" as compared to proprietary tools or technologies.

The Extensible Messaging and Presence Protocol (XMPP) is a general purpose protocol not necessarily limited to instant messaging and presence [XMPP 2004]. XMPP is a revision of the communication portion of the widely deployed Jabber protocol. XMPP is a TCP-based protocol that uses Extensible Markup Language (XML) as the syntax for its protocol elements. XMPP can be used as a client-to-server protocol as well as a server-to-server protocol. The base of the protocol exchange is the XML "stream", effectively a stream of XML data sent from one party to the other which starts with an XML <*stream*> tag and ending with an XML </*stream*> tag. Streams are unidirectional, so communication between two parties requires two separate streams (though they can run over the same full-duplex connection). Within the stream, Requests and Responses are exchanged between the two parties in XML "stanzas", a portion of the stream that has semantic content. The document describes the routing of stanzas from machine to machine through streams. XMPP includes guidelines to ensure that extensions are possible without conflicts or breaking core interoperability. Lack of conflicts is ensured with use of XML namespaces. Interoperability is ensured with careful layering of stanzas of known types, on top of the base stream.

The Internet Engineering Task Force (IETF) has formalized the core XML streaming protocols as an approved instant messaging and presence technology under the name of XMPP, and the XMPP specifications are moving forward rapidly within the IETF's standards process (http://www1.ietf.org/mail-archive/ietf-announce/Current/msg28170.html accessed 29 January 2004)

E. OPEN-STANDARD TECHNOLOGIES AND OPEN-SOURCE SOFTWARE

Open-source software is freely available for any use, including modification and redistribution. The first formal statement of the official Open Source definition appeared in 1997 by Bruce Perens [OSI 2002]. This definition has continued to be refined and maintained by the Open Source Initiative (OSI), a non-profit corporation. [OSI 2004] Developers have a say in how open source are designed and are free to use what works for them, rather than be tied to a particular proprietary package. The plethora of open standards and open source components has shown that this approach is a viable one.

Open source products and tools are based on the premise that the programming source code is freely available to anyone who wishes to read, add to, or even modify and redistribute the computer software code. Thus "free" refers primarily to "freedom to use and modify".

The list of open source libraries used in the AUV Workbench application is given in Table 4.

S/N	Library	Version	Description	Library Files
1.	Apache Ant	1.6.0	Java-based build tool.	ant.jar, optional.jar, xercesImpl.jar, xml-apis.jar
2.	Apache SOAP	2.3.1	Base-64 encoding and decoding.	soap.jar
3.	Apache Xerces	2.5.0	XML parsing.	xmlParserAPIs.jar, xml-apis.jar, xercesImpl.jar
4.	Apache Xalan	2.5.0	XML transformation,	xalan.jar
5.	Batik	1.5.0	A Java based toolkit for apps that want to use images in	batik-awt-util.jar, batik-bridge.jar, batik-css.jar, batik-dom.jar, batik-

S/N	Library	Version	Description	Library Files
			the SVG format for viewing, creation and manipulation.	ext.jar, batik-gvt.jar, batik-parser.jar, batik-script.jar, batik-svg-dom.jar, batik-svggen.jar, batik-swing.jar, batik-util.jar, batik-xml.jar, js.jar
6.	Extensible Java 3D	M8	Display of 3D VRML and X3D models	aviatrix3d-all.jar, gnu-regexp-1.0.8.jar, httpclient.jar, j3d-org-images.jar, j3d-org.jar, Jama.jar, js.jar, JXInput.jar, uri.jar, vlc_uri.jar, vrml97.jar, xj3d-all.jar
7.	Jivesoftware SMACK APIs	1.2.1	XMPP communications.	smack.jar, smackx.jar.
8.	dis-java-vrml	-	Distributed Interactive Simulation.	dis-java-vrml.jar

Table 4. Open source libraries used in the development of AUV Workbench application.

F. PROGRAMMING LANGUAGE AND DEVELOPMENT ENVIRONMENT

The Java Programming Language by Sun Microsystems is the primary language used for this thesis [JDK142]. With Java, numerous commercial and open-source tools, notably Jakarta Apache at http://jakarta.apache.org (accessed February 2004) are available.

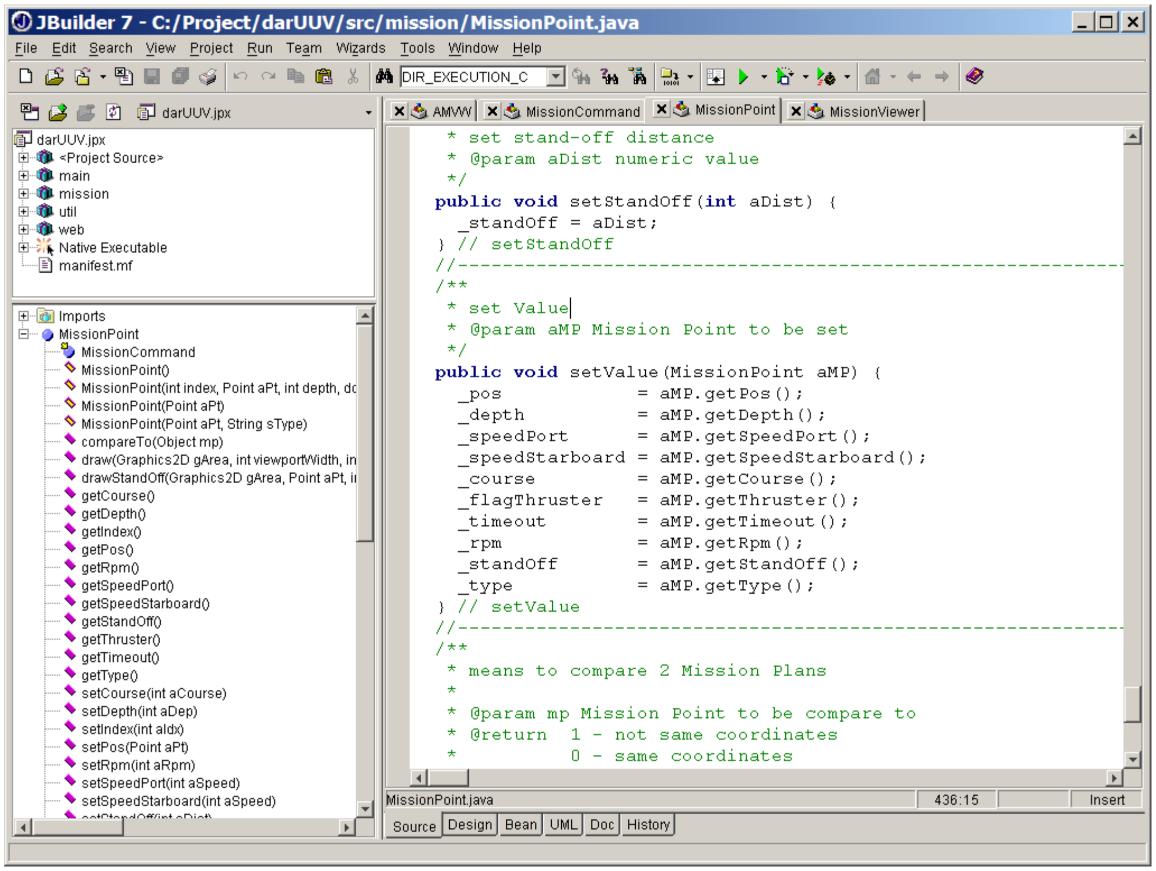
The choice to use Borland's JBuilder 7.0 Enterprise (NPS Education Edition) for development of the AUV Workbench was largely based on the author's familiarity with Borland's Integrated Development Development (IDE) from the use of Borland's Object Pascal, Delphi. The edition of Java used is Java 2 Standard Edition (J2SE) JDK1.4.2. There is no dependency on any particular IDE for development of the NPS AUV Workbench. The NetBeans and Eclipse IDEs are both open source and good no-cost alternatives.

Most IDEs provide tools to easily design a user interface and automatically generate the interface code. This comes at the expense of over-dependence on a particular IDE and likely to pose problems when the user interface needs to be amended on another IDE. Therefore the design and implementation of the AUV Workbench graphical user interface is coded from scratch, instead of using JBuilder's Graphical User Interface (GUI) Designer.

The following sections provide a brief description on some of the IDEs currently available, consisting of both commercial (e.g., Borland JBuilder) and open-source tools such as NetBeans and Eclipse.

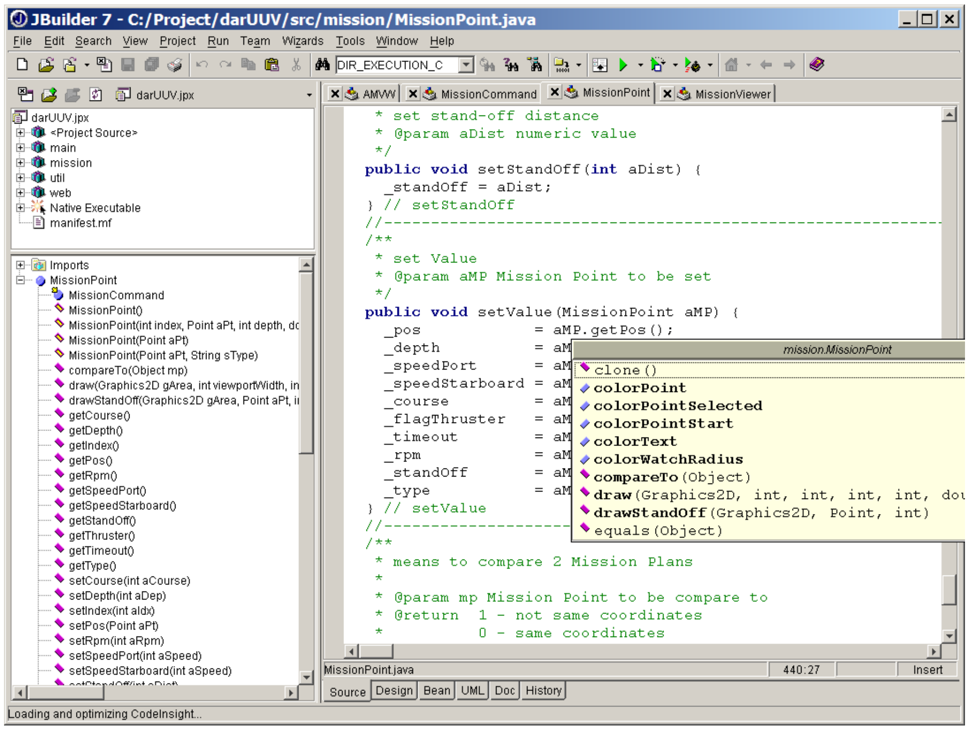
1. JBuilder

JBuilder uses one window to perform most of the development functions: editing, visual designing, navigating, browsing, compiling, debugging, and other operations. This window is called the AppBrowser, and it contains several panes for performing these development functions. The tabbed panes that are available in the content pane depend on what kind of file is selected in the project pane.



```
 * set stand-off distance
 * @param aDist numeric value
 */
public void setStandOff(int aDist) {
  _standOff = aDist;
} // setStandOff
//---------------------------------------------------------------
/**
 * set Value
 * @param aMP Mission Point to be set
 */
public void setValue(MissionPoint aMP) {
  _pos            = aMP.getPos();
  _depth          = aMP.getDepth();
  _speedPort      = aMP.getSpeedPort();
  _speedStarboard = aMP.getSpeedStarboard();
  _course         = aMP.getCourse();
  _flagThruster   = aMP.getThruster();
  _timeout        = aMP.getTimeout();
  _rpm            = aMP.getRpm();
  _standOff       = aMP.getStandOff();
  _type           = aMP.getType();
} // setValue
//---------------------------------------------------------------
/**
 * means to compare 2 Mission Plans
 *
 * @param mp Mission Point to be compare to
 * @return  1 - not same coordinates
 *          0 - same coordinates
```

Figure 8. Borland JBuilder 7.0 application user interface running on Windows XP platform.



JBuilder 7 - C:/Project/darUUV/src/mission/MissionPoint.java
File Edit Search View Project Run Team Wizards Tools Window Help
DIR_EXECUTION_C
darUUV.jpx
AMVW
MissionCommand
MissionPoint
MissionViewer
MissionPoint.java
440:27
Insert
Source
Design
Bean
UML
Doc
History
Loading and optimizing CodeInsight...

```
darUUV.jpx
  <Project Source>
  main
  mission
  util
  web
  Native Executable
  manifest.mf

Imports
MissionPoint
    MissionCommand
    MissionPoint()
    MissionPoint(int index, Point aPt, int depth, dc
    MissionPoint(Point aPt)
    MissionPoint(Point aPt, String sType)
    compareTo(Object mp)
    draw(Graphics2D gArea, int viewportWidth, in
    drawStandOff(Graphics2D gArea, Point aPt, i
    getCourse()
    getDepth()
    getIndex()
    getPos()
    getRpm()
    getSpeedPort()
    getSpeedStarboard()
    getStandOff()
    getThruster()
    getTimeout()
    getType()
    setCourse(int aCourse)
    setDepth(int aDep)
    setIndex(int aIdx)
    setPos(Point aPt)
    setRpm(int aRpm)
    setSpeedPort(int aSpeed)
    setSpeedStarboard(int aSpeed)
```

```
   * set stand-off distance
   * @param aDist numeric value
   */
  public void setStandOff(int aDist) {
    _standOff = aDist;
  } // setStandOff
  //-------------------------------------------------------------
  /**
   * set Value
   * @param aMP Mission Point to be set
   */
  public void setValue(MissionPoint aMP) {
    _pos            = aMP.getPos();
    _depth          = aM
    _speedPort      = aM
    _speedStarboard = aM
    _course         = aM
    _flagThruster   = aM
    _timeout        = aM
    _rpm            = aM
    _standOff       = aM
    _type           = aM
  } // setValue
  //-------------------
  /**
   * means to compare 2 Mission Plans
   *
   * @param mp Mission Point to be compare to
   * @return  1 - not same coordinates
   *          0 - same coordinates
```

```
mission.MissionPoint
clone()
colorPoint
colorPointSelected
colorPointStart
colorText
colorWatchRadius
compareTo(Object)
draw(Graphics2D, int, int, int, int, dou
drawStandOff(Graphics2D, Point, int)
equals(Object)
```

Figure 9. CodeInsight feature running in Borland JBuilder 7.0. This feature displays context-sensitive pop-up windows to facilitate code completion.

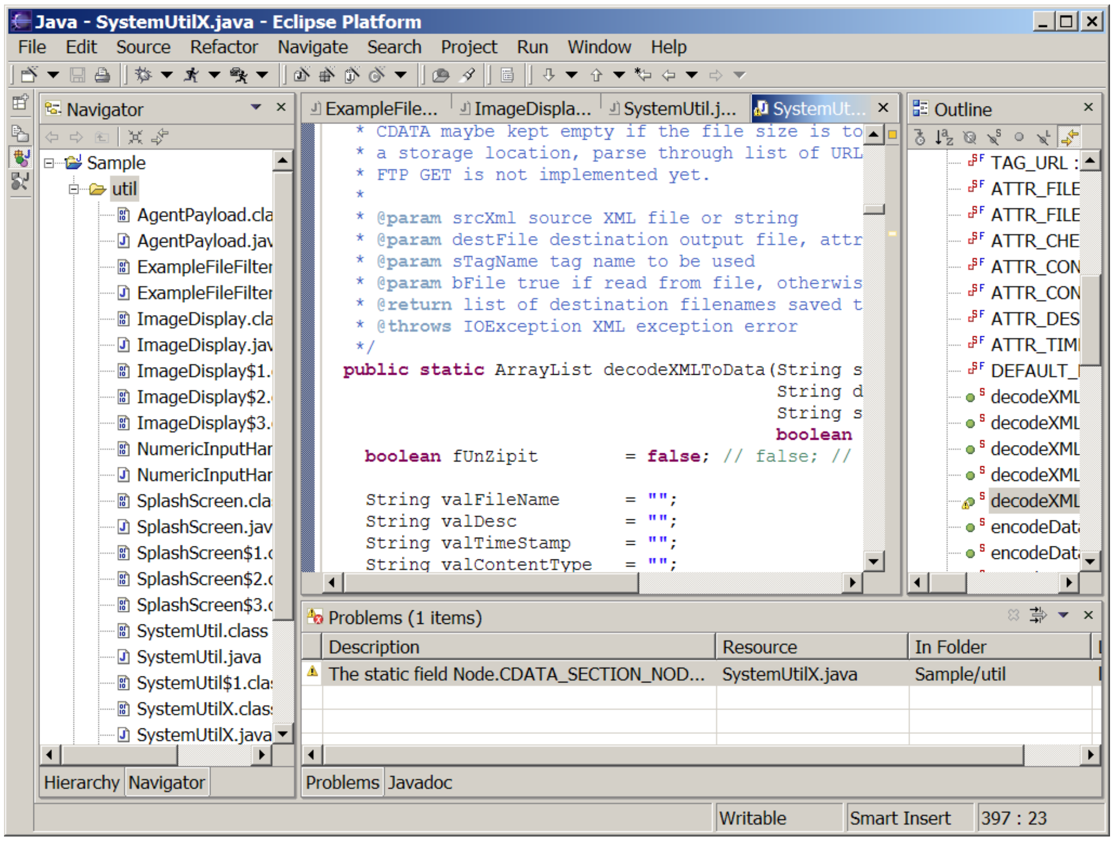
2. Eclipse

Eclipse is an open-source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools. It is composed of three projects, the Eclipse Project, the Eclipse Tools Project and the Eclipse Technology Project (http://www.eclipse.org accessed January 2004.). It is composed of three subprojects: Platform, Java Development Tools (JDT), and Plug-in Development Environment (PDE).

The Eclipse Tools Project provides a focal point for diverse tool builders to ensure the creation of best of breed tools for the Eclipse Platform. The mission of Eclipse Tools Project is to foster the creation of a wide variety of tools for the Eclipse Platform. The Tools project provides single point of coordination for open-source tool developers in order to minimize overlap and duplication, ensure maximum sharing and creation of common components, and promote seamless interoperability between diverse types of tools.

The Eclipse Platform is an open extensible IDE. The Eclipse Platform provides building blocks and a foundation for constructing and running integrated software-development tools. The Eclipse Platform allows tool builders to independently develop tools that seamlessly integrate with other people's tools.

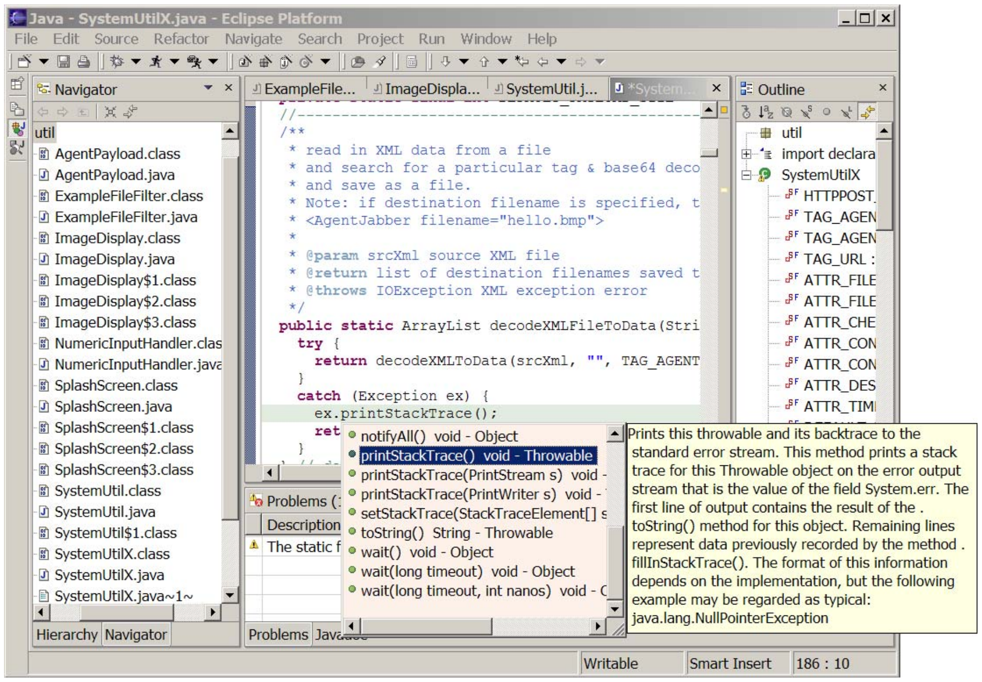
The Eclipse SDK (software developer kit) is the consolidation of the components produced by the three Eclipse Project subprojects (Platform, JDT - Java development tools, and PDE - Plug-in development environment) into a single download.



```
 * CDATA maybe kept empty if the file size is to
 * a storage location, parse through list of URL
 * FTP GET is not implemented yet.
 *
 * @param srcXml source XML file or string
 * @param destFile destination output file, attr
 * @param sTagName tag name to be used
 * @param bFile true if read from file, otherwis
 * @return list of destination filenames saved t
 * @throws IOException XML exception error
 */
public static ArrayList decodeXMLToData(String s
                                        String d
                                        String s
                                        boolean

  boolean fUnZipit        = false; // false; //

  String valFileName      = "";
  String valDesc          = "";
  String valTimeStamp     = "";
  String valContentType   = "";
```

Description	Resource	In Folder
The static field Node.CDATA_SECTION_NOD...	SystemUtilX.java	Sample/util

Figure 10. Eclipse SDK 3.0 Stream Stable Build user interface running on Windows XP platform.



```
//-----------------------------------------------
/**
 * read in XML data from a file
 * and search for a particular tag & base64 deco
 * and save as a file.
 * Note: if destination filename is specified, t
 * <AgentJabber filename="hello.bmp">
 *
 * @param srcXml source XML file
 * @return list of destination filenames saved t
 * @throws IOException XML exception error
 */
public static ArrayList decodeXMLFileToData(Stri
  try {
    return decodeXMLToData(srcXml, "", TAG_AGENT
  }
  catch (Exception ex) {
    ex.printStackTrace();
    ret
  }
```

```
notifyAll()  void - Object
printStackTrace()  void - Throwable
printStackTrace(PrintStream s)  void -
printStackTrace(PrintWriter s)  void -
setStackTrace(StackTraceElement[] s
toString()  String - Throwable
wait()  void - Object
wait(long timeout)  void - Object
wait(long timeout, int nanos)  void - C
```

Prints this throwable and its backtrace to the standard error stream. This method prints a stack trace for this Throwable object on the error output stream that is the value of the field System.err. The first line of output contains the result of the . toString() method for this object. Remaining lines represent data previously recorded by the method . fillInStackTrace(). The format of this information depends on the implementation, but the following example may be regarded as typical: java.lang.NullPointerException

Figure 11. CodeAssist feature running in Eclipse. This feature displays context-sensitive pop-up windows to facilitate code completion.

3. NetBeans

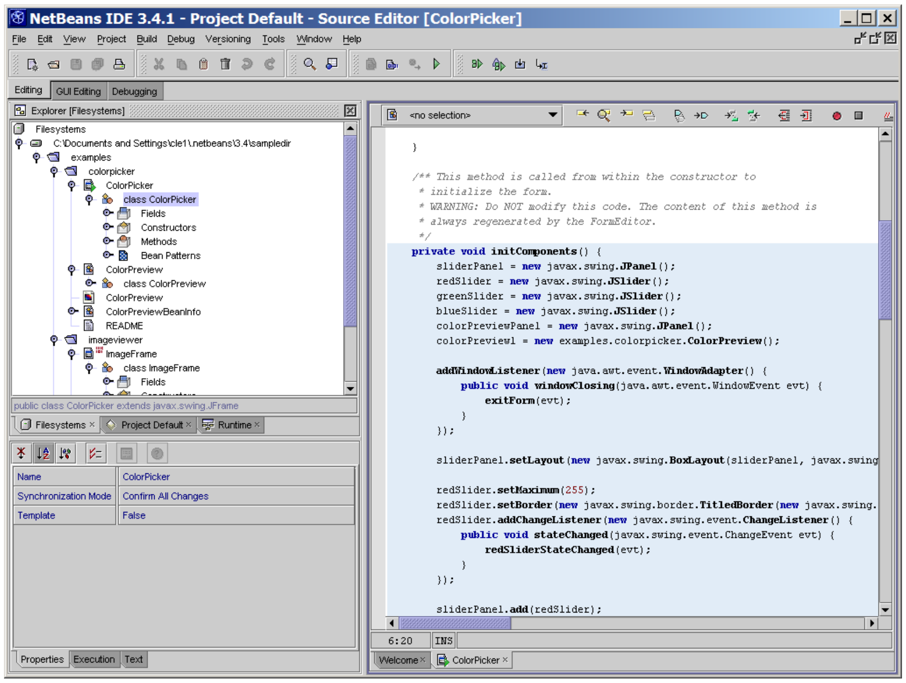
The NetBeans platform is an application runtime - a "generic large desktop application." NetBeans Integrated Development Environment (IDE) comprises the platform and modules such as an editor, tools for working with source code (e.g., Java and C++) and version control. The IDE has advanced syntax highlighting and an error checking code editor that supports Java, C, C++, XML and HTML languages. Some of the features of the platform are (http://www.netbeans.org accessed January 2004):

- User interface management - Windows, menus, toolbars and other presentation components are provided by the Platform. Developers write to a set of abstractions such actions and components, saving time and producing cleaner, more bug-free code. Custom components and behaviors can be written, but for most cases this is not needed.

- Data and presentation management - The NetBeans Platform contains a rich toolset for presentating data to the user and manipulating that data.

- The Editor - Available as an extension to the Platform, applications built on NetBeans can use the NetBeans Editor, a powerful and extensible toolset for building custom editors.
- Setting management - The NetBeans Filesystems infrastructure abstracts file-based data. Files may exist locally or remotely, on FTP or CVS servers or in a database; access to them is transparent to module code that works with files. The Platform can be extended to support new forms of storage. Applications built on NetBeans are Internet-ready.
- The Wizard framework - a toolset for easily building extensible, user-friendly Wizards to guide users through more complex tasks.
- Configuration management - Rather than tediously write code to access remote data and manage and save user-configurable settings, etc., all of this is handled by the Platform. Applications consist of the platform and the logic code important to that application.
- Storage management - An abstraction of file-based data access. "Files" in the NetBeans paradigm may be local files, or exist remotely, for example, on an FTP server, CVS repository or in a database. Where this data is stored is completely transparent to other modules that work with this data.
- Cross-platform - since the Platform is written entirely in the Java language, applications based on it, by their very nature, will run on any operating system with a Java 2 compatible (1.3 or greater) JVM.

The IDE has a dynamic code completion feature for the Java Editor that enables you to type a few characters and then display a list of possible classes, methods, variables, and so on that can be used to automatically complete the expression.


NetBeans IDE 3.4.1 - Project Default - Source Editor [ColorPicker]
File Edit View Project Build Debug Versioning Tools Window Help
Editing GUI Editing Debugging
Explorer [Filesystems]
Filesystems Project Default Runtime
Properties Execution Text
<no selection>
6:20 INS
Welcome ColorPicker

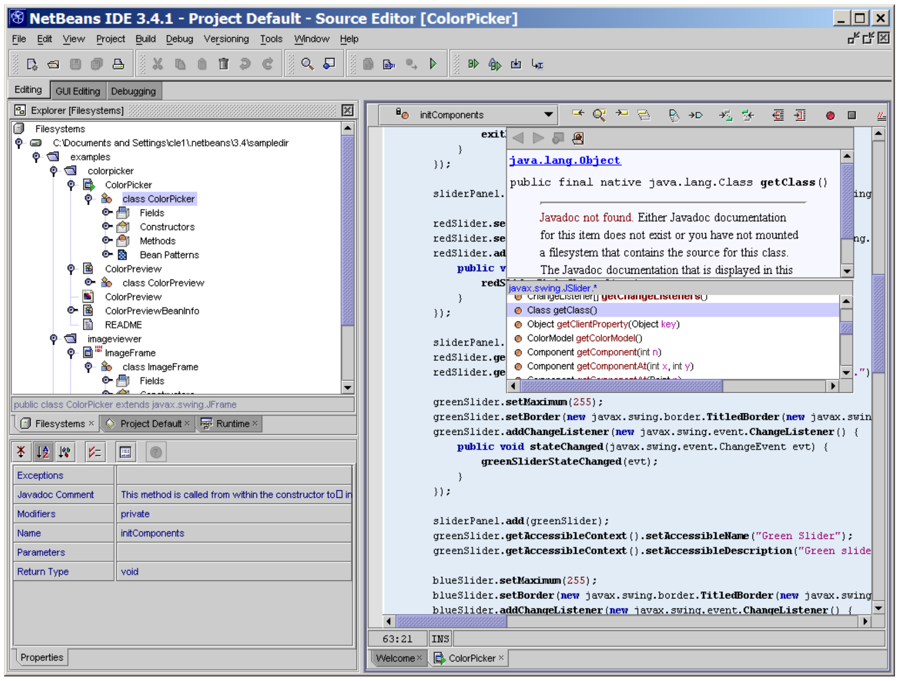
```
Filesystems
C:\Documents and Settings\cle1\.netbeans\3.4\sampledir
  examples
    colorpicker
      ColorPicker
        class ColorPicker
          Fields
          Constructors
          Methods
          Bean Patterns
      ColorPreview
        class ColorPreview
      ColorPreview
      ColorPreviewBeanInfo
      README
    imageviewer
      ImageFrame
        class ImageFrame
          Fields
```

public class ColorPicker extends javax.swing.JFrame

Name	ColorPicker
Synchronization Mode	Confirm All Changes
Template	False

```
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the FormEditor.
     */
    private void initComponents() {
        sliderPanel = new javax.swing.JPanel();
        redSlider = new javax.swing.JSlider();
        greenSlider = new javax.swing.JSlider();
        blueSlider = new javax.swing.JSlider();
        colorPreviewPanel = new javax.swing.JPanel();
        colorPreview1 = new examples.colorpicker.ColorPreview();

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        sliderPanel.setLayout(new javax.swing.BoxLayout(sliderPanel, javax.swing

        redSlider.setMaximum(255);
        redSlider.setBorder(new javax.swing.border.TitledBorder(new javax.swing.
        redSlider.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                redSliderStateChanged(evt);
            }
        });

        sliderPanel.add(redSlider);
```

Figure 12. NetBeans IDE 3.5 user interface running on Windows XP Platform


NetBeans IDE 3.4.1 - Project Default - Source Editor [ColorPicker]
File Edit View Project Build Debug Versioning Tools Window Help
Editing GUI Editing Debugging
Explorer [Filesystems]
Filesystems Project Default Runtime
Properties
initComponents
63:21 INS
Welcome ColorPicker

```
Filesystems
C:\Documents and Settings\cle1\.netbeans\3.4\sampledir
  examples
    colorpicker
      ColorPicker
        class ColorPicker
          Fields
          Constructors
          Methods
          Bean Patterns
      ColorPreview
        class ColorPreview
      ColorPreview
      ColorPreviewBeanInfo
      README
    imageviewer
      ImageFrame
        class ImageFrame
          Fields
```

public class ColorPicker extends javax.swing.JFrame

Exceptions	
Javadoc Comment	This method is called from within the constructor to□ in
Modifiers	private
Name	initComponents
Parameters	
Return Type	void

```
java.lang.Object

public final native java.lang.Class getClass()

Javadoc not found. Either Javadoc documentation
for this item does not exist or you have not mounted
a filesystem that contains the source for this class.
The Javadoc documentation that is displayed in this
```

```
javax.swing.JSlider.*
Class getClass()
Object getClientProperty(Object key)
ColorModel getColorModel()
Component getComponent(int n)
Component getComponentAt(int x, int y)
```

```
                exit
            }
        });

        sliderPanel.

        redSlider.se
        redSlider.se
        redSlider.ad
            public v
                redS
            }
        });

        sliderPanel.
        redSlider.ge
        redSlider.ge

        greenSlider.setMaximum(255);
        greenSlider.setBorder(new javax.swing.border.TitledBorder(new javax.swin
        greenSlider.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                greenSliderStateChanged(evt);
            }
        });

        sliderPanel.add(greenSlider);
        greenSlider.getAccessibleContext().setAccessibleName("Green Slider");
        greenSlider.getAccessibleContext().setAccessibleDescription("Green slide

        blueSlider.setMaximum(255);
        blueSlider.setBorder(new javax.swing.border.TitledBorder(new javax.swing
        blueSlider.addChangeListener(new javax.swing.event.ChangeListener() {
```

Figure 13. Code completion feature running in NetBeans IDE 3.5. This feature displays context-sensitive pop-up windows to facilitate code completion.

G. NPS ARIES AUTONOMOUS UNDERWATER VEHICLES (AUV)

1. Introduction

The Naval Postgraduate School Center for AUV Research has been building, operating, and researching autonomous underwater vehicles (AUVs) since 1987. Each new generation of vehicles have substantially increased operational capabilities and are much more robust and sophisticated in terms of hardware and computer software. These vehicles have also moved from operating in swimming pool environments to the open ocean [Oceans 2000]. The latest NPS vehicle is named Acoustic Radio Interactive Exploratory Server (ARIES). This vehicle is a student-research test bed for shallow-water minefield-mapping missions, operating in the literal ocean. Currently the vehicle operates regularly in Monterey Bay [Grunesien 2002].

2. Dimensions and Endurance

The vehicle weighs 225 Kg and measures approximately 3 m long wide and 0.25 m high. The hull is constructed of 6.35 mm thick type 6061 aluminum and forms the main pressure vessel that house all electronics, computers and batteries. A flooded fiberglass nose is used to house the external sensors, key-controlled power “on/off” switches and status indicators. ARIES is capable of a top speed of 3.5 knots and is powered by six 12-volt rechargeable lead-acid batteries. Vehicle endurance is approximately 4 hours at top speed, with 20 hours endurance under “hotel load” only. The ARIES is primarily designed for shallow-water operations and can operate safely down to depths of 30 meters [Oceans 2000].

3. Propulsion and Motion Control Systems

Main propulsion is achieved using twin ½ Hp electric drive thrusters located at the stern. During normal submerged flight, heading and depth are controlled using upper bow and stern rudders plus a set of bow planes and stern planes. Since the control fins are ineffective during slow (or zero) forward-speed maneuvers, vertical and lateral cross-body thrusters are used to control surge, sway, heave, pitch and yaw motions [Oceans 2000].

4. Navigation Sensors

The sensor suite used for navigations includes a 1200 kHz Instruments (RDI) Navigator Doppler Velocimeter Log (DVL) that also contains a TCM2 magnetic compass. This instrument measures the vehicle ground speed, altitude, and magnetic heading. Angular rates and accelerations are measured using a Systron Donner 3-axis Motion pak IMU. While surfaced, Global Positioning System (GPS) inputs is provided by a carrierphase differential GPS (DGPS CP) system, available during surfaced operation to correct any navigational errors accumulated during the submerged phases of a mission [Oceans 2000].

5. Sonar and Video Sensors

Tritech ST725 scanning sonar and an ST1000 profiling sonar is used for obstacle avoidance and target acquisition/reacquisition. The sonar heads can scan continuously through 360 degree of rotation or swept through a predefined angular sector. A fixed-focus wide-angle video camera is located in the nose and is connected to a DVC recorder. The computer is interfaced to the recorder that controls on/off and start/stop record functions. While recording images, data for date, time, vehicle position, depth and altitude is superimposed on the video image [Oceans 2000].

6. Vehicle/Operator Communications

Radio modems are used for high bandwidth command, control and system monitoring while the vehicle is deployed and surfaced. While submerged, an acoustic modem is used for low-bandwidth communications. In the laboratory environment, a 10 Mbps thin-wire Ethernet connection is used for software development and mission data upload and download [Oceans 2000].

7. Computer Hardware Architecture

The dual-computer system unit measures approximately 28 x 20 x 20 cm. It consists of two Ampro Little Board 166 MHz Pentium computers with 64 MB RAM, four serial ports, a network adapter and a 2.5 GB hard drive each. Two DC/DC voltage converters for powering both computer systems and peripherals are integrated into the computer package. The entire computer system draws a nominal 48 Watts. Both systems use TCP/IP sockets over thinwire Ethernet for inter-processor communications as well as connections to an external LAN. The sensor data-collection computer is designated QNXT. The second is named QNXE and executes the various auto-pilots for servo-level control [Oceans 2000].

8. Computer Software Architecture

The ARIES AUV uses a tri-level software architecture called the Rational Behavior Model (RBM). RBM divides the responsibilities into areas of open-ended strategic planning, soft-real-time tactical analysis and hard-real-time execution-level control. The RBM architecture is modeled after a manned submarine operational structure. The correspondence between the three levels and a submarine crew is shown in Figure 14 below.

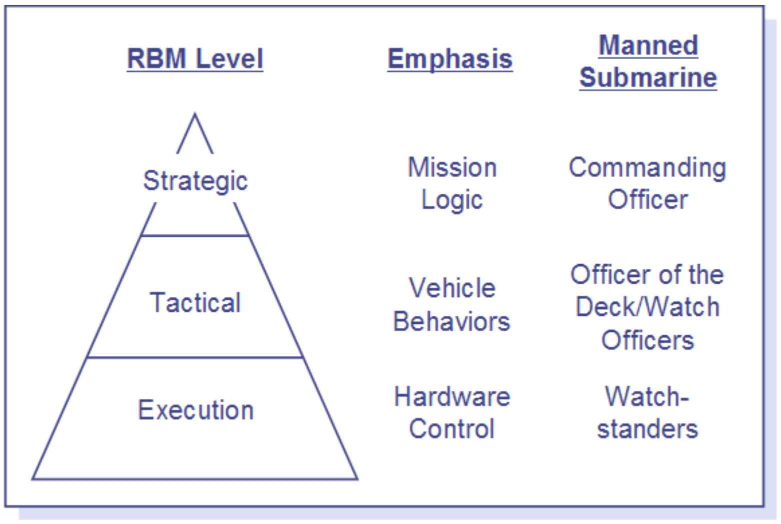


Figure 14. Relational Behavior Model tri-level architecture hierarchy with level emphasis and submarine equivalent listed [Holden 1995].

This figure represents the tri-level software hierarchy with level emphasis and submarine equivalent listed. The Execution Level assures the interface between hardware and software. Its tasks are to maintain the physical and operational stability of the vehicle, to control the individual devices and to provide data to the tactical level. These tasks are currently performed by on-board host QNXS computer. The Tactical Level provides a software level that interfaces with both the Execution Level and the Strategic Level. Its chores are to give to the Strategic level indications of vehicle state, completed tasks and execution level commands. The Tactical level selects the tasks needed to reach the goal imposed by the Strategic level. It operates in terms of discrete events.

The Strategic Level controls the completion of the mission goals. The mission specifications are inside this level.

H. RELATED RESEARCH

1. History and Contributors

The AUV Workbench is the result of the combined efforts of several past and present NPS students and faculty. Adrien Gruneisen and Yann Henriet [Grunesien 2002] developed the first version of the workbench based on the dissertation research of Don Brutzman [Brutzman 1994]. It executes AUV missions while providing the user with a "close-up" view of the vehicle so the vehicle dynamics can be observed.

Doug Horner added support for a non-validating XML-based mission script, an obstacle avoidance algorithm, and support of mission planning using plain text format. The original vehicle execution was re-written using Java network communications. Of note, the XML-based mission script format has been superseded by the Hawkins and Van Leuvan thesis effort in 2003 [Hawkins 2003] and the Ayala thesis on AUV Java execution using Distributed Interactive Simulation (DIS) supersedes the older vehicle execution [Ayala 2002]. The obstacle avoidance module does not compute the path dynamically. It generates a path based on a list of known obstacles and the preloaded AUV mission script. The initial efforts were to implement a simple standalone application for pre-mission visualization and as a quick prototype for proof-of-concept. Therefore there was no network connectivity (e.g., through IEEE Distributed Interactive Simulation DIS protocol) and no collaboration tools were introduced.

I. SUMMARY

The NPS AUV Workbench integrates years of research work by students and faculty. To take the AUV Workbench to the next step, it is necessary to streamline these efforts and employ the best practices of software development. It is through such an approach that important knowledge can be retained and continued research and development can be promoted.

THIS PAGE INTENTIONALLY LEFT BLANK

III. AUV WORKBENCH

A. INTRODUCTION

The AUV Workbench is a common mission planning and analysis tool for AUVs. It supports physics-based AUV modeling and visualization of vehicle behavior and sensors in all mission phases: pre-mission, post-mission and ongoing mission visualization. The AUV workbench consists of four main modules. These modules communicate with each other either directly or over the network for required interaction. The individual modules are responsible for four distinct functions: mission execution; virtual world dynamics modeling and feedback; mission planning and generation; and 2D/3D mission visualization. This chapter provides details regarding the design and implementation of the modules developed under this thesis, namely mission planning, XML-based mission script and the general Workbench interface. The topics on mission execution and virtual world modeling and feedback are also explained. Two supporting modules are included to facilitate information exchange among human operators as well as agents.

B. DESIGN RATIONALE

1. Graphical User Interface (GUI)

The Workbench user interface is divided into four distinct sections. Text or XML-based mission scripts are loaded as part of the Mission panel, in the upper left pane. Clicking on `List` and `Text` tabbed pages toggle between the various modes of the mission script. The Mission Planner and three-dimensional visualization displays provides the viewing panel on the upper right pane. The modules, Execution and Dynamics, to model the robot and its virtual environment are found at the bottom of the Workbench window. By default, the application toolbar is located on the right side. It allows the addition of custom applications to be added and launched in a separate process. Of note, the toolbar is both dock-able and float-able. The user can choose to dock the toolbar on any side of the Workbench application, or keep it floating.

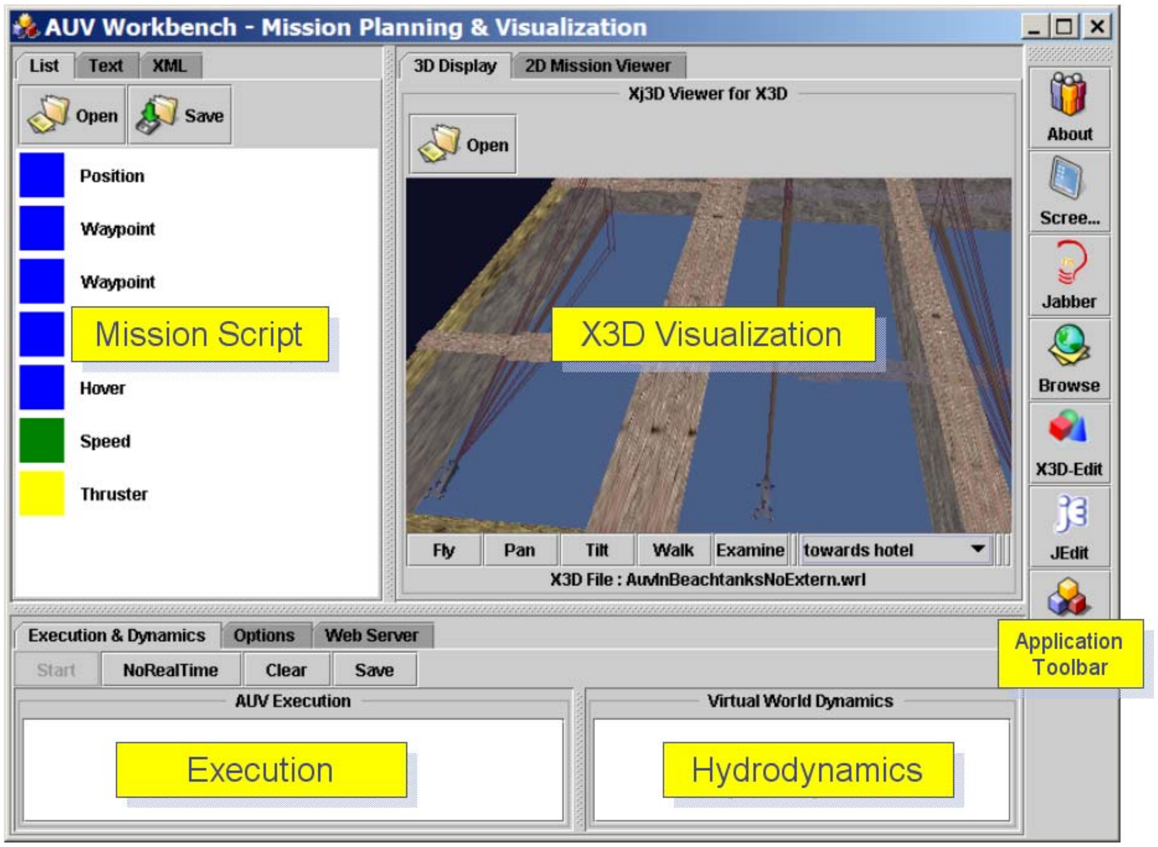


Figure 15. AUV Workbench application user interface.

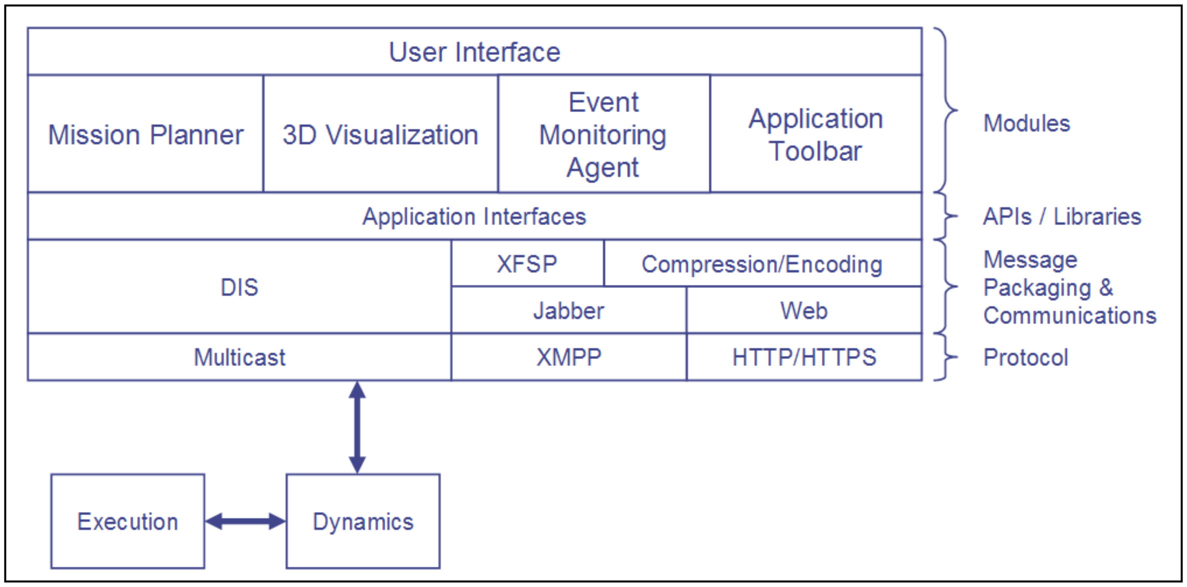


Figure 16. List of modules and libraries required to build the AUV Workbench application.

2. Project Structure

The AUV Workbench is Java-based and was implemented using a componentized framework. The project structure is shown in Figure 17. At the top level, the core directories are */bin, /lib, /execution, /dynamics, /Models, /Scripts, /dataweb* and */dataim*. The Java packages and classes that make-up the Workbench are kept in /bin directory. The list of required libraries such *Apache Xerces* for XML parsing are stored in /*lib*. The robot control and virtual environment modules are found in /execution and /*dynamics* respectively. To facilitate the user to get started quickly, sample models and mission scripts are distributed in /*Models* and /*Scripts* directories. */dataweb* and /*dataim* store files that are used by the web server and Jabber instant messaging modules.

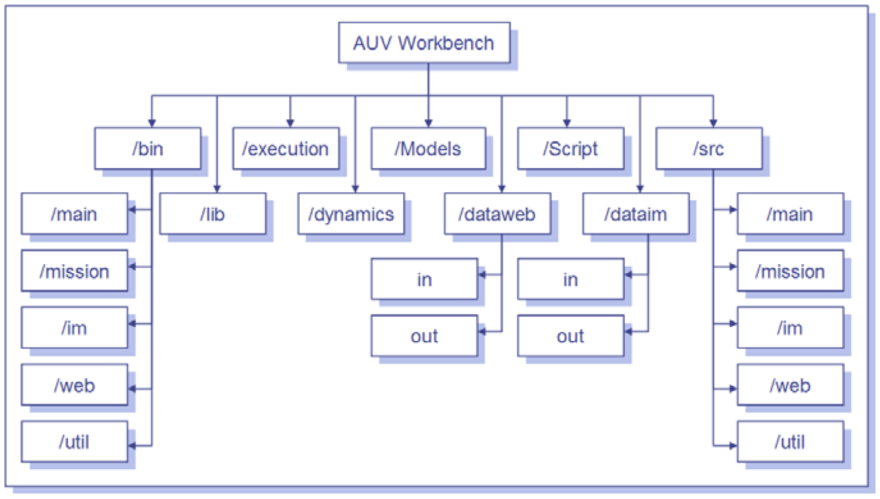


Figure 17. AUV Workbench project directory structure.

3. Source Code and Runtime Package Structure

The directories in Figure 18 illustrate the directory structure of the AUV Workbench Java source and runtime packages.

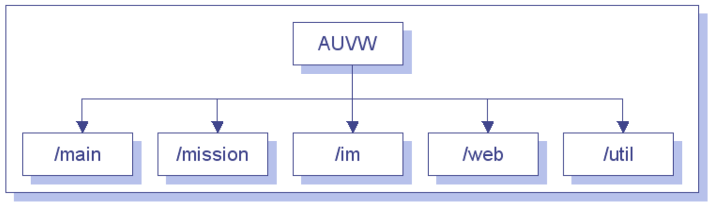


Figure 18. AUV Workbench application (AUVW) Java source and binary directory structure.

This setup provides ease of development and subsequent maintenance of the different modules. */main* contains the source code of the main user interface and the 3D visualization. It is responsible for the rendering of the entire user interface including the placements of the user interfaces for the various modules. The two-dimensional mission planner module is placed in */mission*. Jabber instant messaging and web server modules are placed in */im* and */web* respectively. Common utilities and procedures are kept in /util.

S/N	Name	Description
1.	main	Main user interface and 3D Visualization module.
2.	mission	Two-dimensional mission planner module.
3.	im	Jabber Instant Messaging and XTC Event Monitor modules.
4.	web	Web server
5.	util	Common utilities.

Table 5. A summary of AUV Workbench packages.

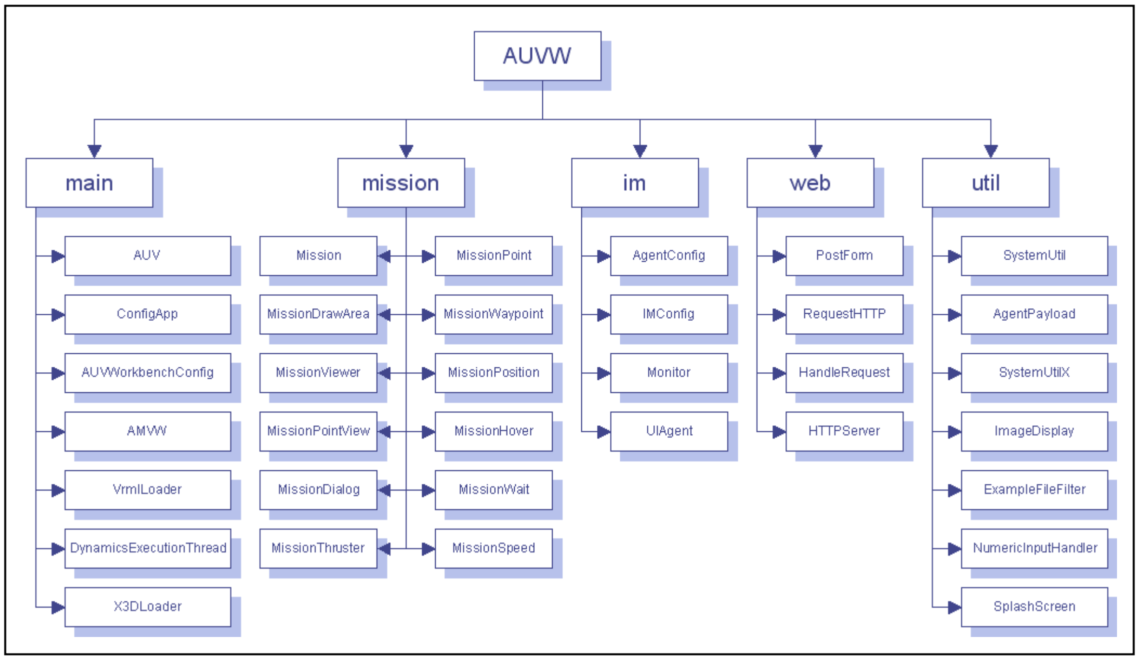


Figure 19. Overview of AUV Workbench classes.

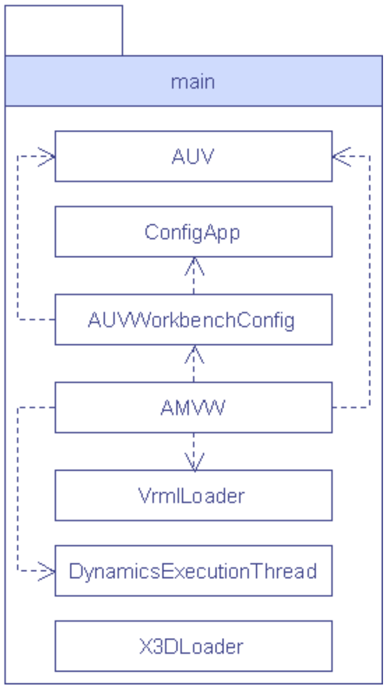


Figure 20. “Main” module package.

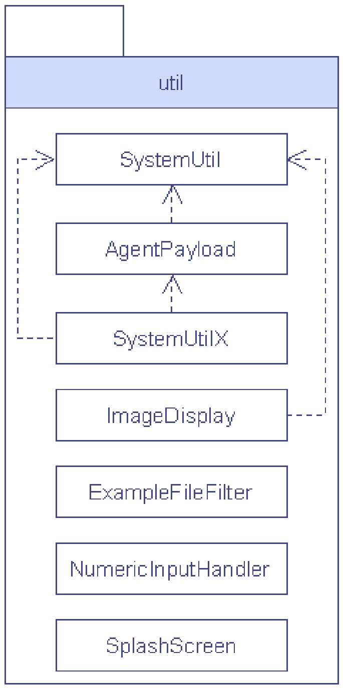


Figure 21. “Util” module package.

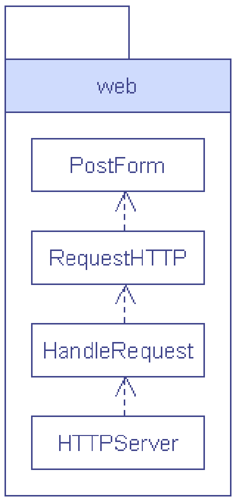


Figure 22. “Web” module package.

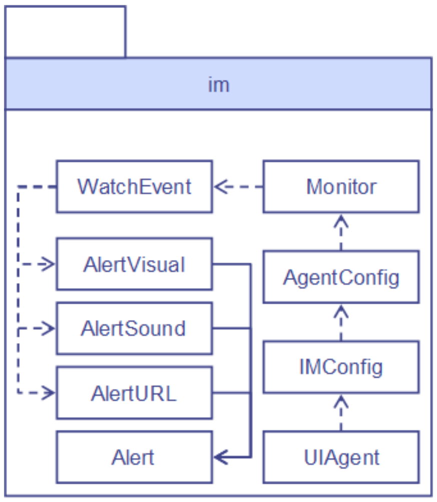


Figure 23. “Im” module package.

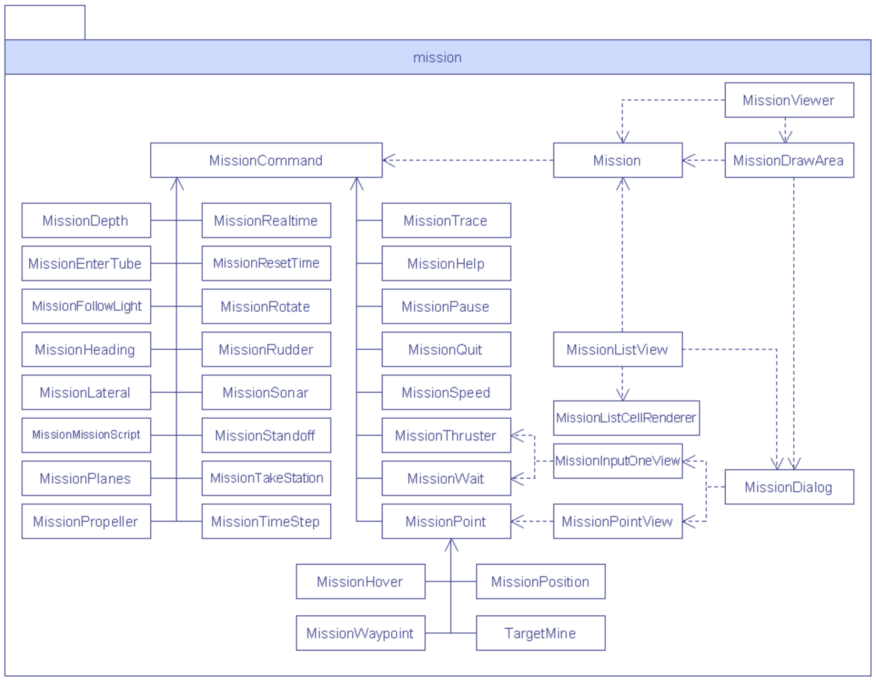


Figure 24. “Mission” module package.

4. Configuration File

Although the componentized framework works well for developers, day-to-day users of the Workbench require something simpler so that they can make changes to the system easily and move on to their actual work. Therefore an XML-based configuration file has been introduced. This file is located in the same directory as the application executeable. Adding a new tool is as simple as opening the configuration file, *AUVWorkbenchConfiguration.xml*, adding a new entry under the *Application* stanza section and re-starting the Workbench. Details on adding new tools to the application toolbar will be discussed in a subsequent section.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  <head>
   <meta name="filename" content="AUVWorkbenchConfiguration.xml" />
   <meta name="authors"  content="Daryl Lee, Duane Davis, Don Brutzman,
US Naval Postgraduate School, Monterey, CA" />
   <meta name="created"  content="15 February 2004" />
   <meta name="revised"  content="15 February 2004" />
   <meta name="description"
         content="This file contains the AUV Workbench configuration/>
   <meta name="url"
content="C:/auv/Workbench/Scripts/AuvCommandLanguage.xslt" />
   <meta name="document summary" content="A valid document will have a
```

```
AUVWorkBench command root element. A AUVWorkBench element can contain 1
General, 1 Execution, 1 Hydrodynamics 1 EventMonitor and 1
PluginManager"/>
</head>
 -->
<AUVWorkBench>
   <General>
       <Models>../Models/</Models>
       <Scripts>../Scripts/</Scripts>
       <Application name="Jabber" tooltip="Instant Messaging Client"
image="image/jabber.gif" show="true">
          <Command>C:/Program Files/RhymBox/RhymBox.exe</Command>
          <Command>D:/Program Files/RhymBox/RhymBox.exe</Command>
          <Command>C:/Program Files/IM/RhymBox/RhymBox.exe</Command>
       </Application>
       <Application name="Browse" tooltip="Web Browser"
image="image/browser.gif" show="true" content-type="text/html">
          <Command>C:/Program
Files/mozilla.org/Mozilla/mozilla.exe</Command>
          <Command>D:/Program
Files/mozilla.org/Mozilla/mozilla.exe</Command>
          <Command>C:/Program Files/Internet
Explorer/IEXPLORE.EXE</Command>
          <Command>D:/Program Files/Internet
Explorer/IEXPLORE.EXE</Command>
       </Application>
       <Application name="X3D-Edit" tooltip="X3D Editor"
image="image/x3d.gif" show="true" content-type="model/x3d">
          <Command>C:/www.web3d.org/TaskGroups/x3d/translation/X3D-
Edit.bat</Command>
          <Command>D:/www.web3d.org/TaskGroups/x3d/translation/X3D-
Edit.bat</Command>
          <Command>C:/www.web3d.org/TaskGroups/x3d/translation/X3D-
Edit-English.bat</Command>
       </Application>
       <Application name="JEdit" tooltip="JEdit"
image="image/jedit.gif" show="true" content-type="text/x-java">
          <Command>C:/Program Files/jEdit 4.1/jedit.exe</Command>
          <Command>D:/Program Files/jEdit 4.1/jedit.exe</Command>
       </Application>
       <Application name="ADS" tooltip="AUV Data Server"
image="image/3cubes.gif" show="true">
          <Command>C:/auv/ADS/AuvDataServer.bat</Command>
          <Command>D:/auv/ADS/AuvDataServer.bat</Command>
       </Application>
       <Application name="NotePad" tooltip="Windows Notepad"
image="image/note.gif" show="false" content-type="text/plain,
text/xml">
          <Command>C:/windows/NOTEPAD.EXE</Command>
       </Application>
       <Application name="Picture" tooltip="Windows Fax and Viewer"
image="image/graphics.gif" show="false" content-type="image/bmp,
image/gif">
          <Command>C:/windows/System32/mspaint.exe</Command>
          <Command>D:/windows/System32/mspaint.exe</Command>
       </Application>
       <Application name="SVGVRML" tooltip="Display VRML and SVG"
image="image/SVG.gif" show="false" content-type="model/vrml,
image/svg+xml">
          <Command>C:/Program Files/Internet
Explorer/IEXPLORE.EXE</Command>
          <Command>D:/Program Files/Internet
Explorer/IEXPLORE.EXE</Command>
       </Application>
       <Webserver docroot="../dataweb/" port="80" autostart="false"
upload="../dataweb/in/" />
       <Jabber dirIn="../dataim/in/" dirOut="../dataim/out/"
domain="surfaris.cs.nps.navy.mil" port="5222" username="lee"
nickname="WorkBenchDaryl" resource="Work"
jid="savage@conference.xchat.movesinstitute.org"/>
    </General>
    <Execution>
       <ExecutionJava>../Java
execution/classes/Execution</ExecutionJava>
       <ExecutionC>../execution/execution.exe</ExecutionC>
    </Execution>
    <Hydrodynamics>
    <Dynamics>../dynamics/classes/dynamics</Dynamics>
       <AUV number="1" multicastGroup="224.2.181.145"
multicastPort="62040" ttl="15" applicationID="0" siteID="0"
entityID="1" desc="AUV in Beach Tank 1" />
       <AUV number="2" multicastGroup="224.2.181.145"
multicastPort="62040" ttl="15" applicationID="0" siteID="0"
entityID="2" desc="AUV in Beach Tank 2" />
       <AUV number="3" multicastGroup="224.2.181.145"
multicastPort="62040" ttl="15" applicationID="1" siteID="1"
entityID="36" desc="AUV in Beach Tank 3" />
    </Hydrodynamics>
    <EventMonitor>
       <MonitorDefault keywordSubject="mine " keywordBody="nice, mine">
          <WatchEvent expr="^.*(?i)MINE[s|S]? .*[ (]{1,2}(\d*),[
]{0,2}(\d*),[ ]{0,2}(\d*)[ ).]?+">
             <Alert type="visual" src="image/mine.gif"/>
             <Alert type="sound" src="sound/alert.wav"/>
             <Alert type="url" src="C:/auv/Workbench/doc/index.htm"/>
          </WatchEvent>
          <WatchEvent expr="^.*(?i)SHIP[s|S]? .*[ (]{1,2}(\d*),[
]{0,2}(\d*),[ ]{0,2}(\d*)[ ).]?+">
             <Alert type="visual" src="image/ship.gif"/>
          </WatchEvent>
          <WatchEvent expr="^.*(?i)LOCATION[s|S]? .*[ (]{1,2}(\d*),[
]{0,2}(\d*),[ ]{0,2}(\d*)[ ).]?+" alert=""/>
       </MonitorDefault>
       <Monitor jid="savage@conference.xchat.movesinstitute.org"
desc="" datetimeStart="" dateTimeEnd="" keywordSubject=" location"/>
       <Monitor jid="auvrobot@surfaris.cs.nps.navy.mil" desc=""
datetimeStart="" datetimeEnd=""/>
    </EventMonitor>
</AUVWorkBench>
```

Figure 25. Sample AUV Workbench configuration file.

S/N	Name	Type	Description
1.	AUVWorkBench	Element	Root.
2.	General	Element	Application configurations.
3.	Models	Element	Directory location of 3D models.
4.	Scripts	Element	Directory location of mission scripts.
5.	Execution	Element	Not used.
6.	ExecutionJava	Attribute	Location of Java class for execution application.
7.	ExecutionC	Attribute	Location of C program for execution application.
8.	Hydrodynamics	Element	Not used.
9.	AUV	Element	Not used.
10.	multicastGroup	Attribute	Multicast address. Not used.
11.	multicastPort	Attribute	Multicast port no. Not used.
12.	ttl	Attribute	Multicast packet time-to-live. Not used.
13.	applicationID	Attribute	DIS packet application ID. Not used.
14.	siteID	Attribute	DIS packet site ID. Not used.
15.	entityID	Attribute	DIS packet entity ID. Not used.
16.	desc	Attribute	Description. Not used.

Table 6. XML tagset to define the AUV Workbench configuration.

5. ANT – JAVA-based Build Tool

Ant is a Java-based build tool. In theory, it is kind of like Make, without Make's wrinkles and with the full portability of pure Java code. According to Ant's original author, James Duncan Davidson, the name is an acronym for "Another Neat Tool". Ant builds projects specified by an XML build file. The Build file defines Build targets and Build tasks. For example, a build file might contain separate targets for building a project and generating Javadoc. The individual targets or the default target for the project can be executed using the Ant build file (http://ant.apache.org accessed January 2004). The `build.xml` for AUV Workbench is given in Figure 26.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!-- ANT Build Script for the AUV Workbench Project -->
<project name="AUVWorkbench" default="bin" basedir=".">

  <!-- ############# Project Standard Properties ########### -->
```

```
<property name="project.name"    value="AUVWorkbench" />
<property name="project.version" value="0.1" />

<!-- Java source and package directory -->
<property name="src.dir"         value="${basedir}/src" />
<property name="src.main.dir"    value="${src.dir}/main" />
<property name="src.mission.dir" value="${src.dir}/mission" />
<property name="src.im.dir"      value="${src.dir}/im" />
<property name="src.web.dir"     value="${src.dir}/web" />
<property name="src.util.dir"    value="${src.dir}/util" />

<!-- Library dependencies -->
<property name="lib.dir"         value="${basedir}/lib" />

<!-- Java compiled and package directory -->
<property name="build.dir"         value="${basedir}/bin" />
<property name="build.main.dir"    value="${build.dir}/main" />
<property name="build.mission.dir" value="${build.dir}/mission" />
<property name="build.im.dir"      value="${build.dir}/im" />
<property name="build.web.dir"     value="${build.dir}/web" />
<property name="build.util.dir"    value="${build.dir}/util" />
<property name="build.image.dir"   value="${build.dir}/image" />

<!-- distribution directory is the same as bin for the moment -->
<property name="dist.dir"           value="${basedir}/bin" />
<property name="dist.jar.file"
    value="${dist.dir}/${project.name}-${project.version}.jar" />
<property name="manifest.file"
          value="${build.dir}/META-INF/manifest.mf" />

<!-- Java documentation directory -->
<property name="javadocs.dir"      value="${basedir}/javadocs" />

<!-- Javadocs ZIP file -->
<property name="javadocs.file"
 value="${dist.dir}/${project.name}-${project.version}-javadocs.zip" />

<!-- include dependent libraries to classpath -->
<path id="build.classpath">
  <fileset dir="${lib.dir}">
    <include name="*.jar" />
    <include name="*.zip" />
  </fileset>
</path>
<!-- ################## Project Build ################## -->
<!-- ################## "clean" command ################## -->
<!-- Clean-up existing files and directories -->
<target name="clean">
  <!-- remove compiled packages -->
  <delete dir="${build.main.dir}" />
  <delete dir="${build.mission.dir}" />
  <delete dir="${build.im.dir}" />
  <delete dir="${build.web.dir}" />
  <delete dir="${build.util.dir}" />
```

```
    <!-- remove JAR file -->
    <delete file="${dist.jar.file}" />

    <!-- remove JavaDocs -->
    <delete dir="${javadocs.dir}" />
  </target>

  <!-- ################# "prepare" command ################# -->
  <!-- Create the destination directories -->
  <target name="prepare" depends="clean">
    <!-- create packages directory -->
    <mkdir dir="${build.main.dir}" />
    <mkdir dir="${build.mission.dir}" />
    <mkdir dir="${build.im.dir}" />
    <mkdir dir="${build.web.dir}" />
    <mkdir dir="${build.util.dir}" />
    <mkdir  dir="${javadocs.dir}" />
  </target>

  <!-- ################# "command" command ################# -->
  <target name="compile" depends="prepare"
          description="compile all the source codes">
    <javac srcdir="${basedir}/src"
           destdir="${build.dir}" deprecation="true">
      <classpath refid="build.classpath" />
    </javac>
  </target>

  <!-- ########### "dist" command to generate JAR ########## -->
  <target name="dist" depends="compile">
    <jar jarfile="${dist.jar.file}"
         basedir="${build.dir}" manifest="${manifest.file}">
    </jar>
      </target>

  <!-- ################# "javadoc" command ################# -->
  <target name="javadoc" depends="compile">
    <javadoc destdir="${javadocs.dir}"
             windowtitle="${project.name}
                          Class Library (version ${project.version})"
             overview="${basedir}/src/overview.htm">
      <classpath refid="build.classpath" />
      <packageset dir="${src.dir}" defaultexcludes="yes">
      </packageset>
    </javadoc>

    <!-- create a zip file for the javadocs in
         distribution directory -->
    <zip zipfile="${javadocs.file}">
      <zipfileset dir="${javadocs.dir}"
                  prefix="${project.name}-${project.version}-javadocs" />
    </zip>
  </target>

  <!-- ################### "all" command ################### -->
  <target name="all"
```

```
           depends="dist,javadoc"
           description="Compiles the source, builds the jar files,
                        generates the Javadoc HTML pages and creates
                        distribution files (.zip).">
  </target>
</project>
```

Figure 26. AUV Workbench ANT build.xml used to compile and build the application.

Here is a detailed examination of the build.xml file to explain what it does:

- *project*: includes a project name, the default target to run if none of the other individual targets are run, and the location of the base directory (*/bin*).
- *properties*: Ant targets and tasks are typically "property-aware". Properties are also used to pass parameters to tasks without overriding the existing properties in the build file. To get the value of a property, use "*${<property name>}*" syntax.
- *clean* target: deletes existing compiled packages' directories and the project JAR file.
- *prepare* target: creates the package directories for the compiled classes.
- *compile* target: initiates the *clean* target first (using the *depends* keyword), which in turn initiates *clean* target, then compiles the Java source files and puts the generated .class files in the build directory.
- *dist* target: initiates the *compile* target first and creates a JAR file in that directory.
- *javadoc* target: creates the JavaDoc for the project and also generate a compressed copy of the Java documentations.
- *zip* target: Compress all the project dependencies into a single ZIP file.
- *all* target: initiates *dist*, *javadoc* and *zip* targets.
- To activate the individual targets, use "ant <*target name*>"; e.g., "*ant compile*". By default, Ant looks for "*build.xml*". To specify a different Build file name, use "*ant –buildfile <Build filename> <target name>*"

```
JAVA_HOME=C:\Application\ j2sdk1.4.2
ANT_HOME=C:\apache-ant-1.6.0
Buildfile: build.xml

clean:
   [delete] Deleting directory C:\Project\darUUV-ant\bin\main
   [delete] Deleting directory C:\Project\darUUV-ant\bin\mission
   [delete] Deleting directory C:\Project\darUUV-ant\bin\im
   [delete] Deleting directory C:\Project\darUUV-ant\bin\web
   [delete] Deleting directory C:\Project\darUUV-ant\bin\util
   [delete] Deleting: C:\Project\darUUV-ant\bin\AUVWorkbench-0.1.jar
   [delete] Deleting directory C:\Project\darUUV-ant\javadocs

prepare:
    [mkdir] Created dir: C:\Project\darUUV-ant\bin\main
    [mkdir] Created dir: C:\Project\darUUV-ant\bin\mission
    [mkdir] Created dir: C:\Project\darUUV-ant\bin\im
    [mkdir] Created dir: C:\Project\darUUV-ant\bin\web
    [mkdir] Created dir: C:\Project\darUUV-ant\bin\util
    [mkdir] Created dir: C:\Project\darUUV-ant\javadocs

compile:
    [javac] Compiling 44 source files to C:\Project\darUUV-ant\bin

dist:
      [jar] Building jar: C:\Project\darUUV-ant\bin\AUVWorkbench-
0.1.jar

BUILD SUCCESSFUL
Total time: 12 seconds
```
Figure 27. Output from “ant dist” command running the AUV Workbench “build.xml” file.

C. MISSION PLANNING

1. Overview

The AUV Workbench supports both a simple, text-based mission script as well as the use of XML-based mission scripts. This section presents the details of the XML-based mission script and the tools to author them.

2. AUV XML-based Mission Control Script

The XML AUV command language is defined using XML schema [Hawkins 2003]. In general the command language enables the explicit declaration of an entire mission using execution-level commands, the ad-hoc definition of a mission by providing individual commands asynchronously (in the form of individual single command element documents), mission data archiving (telemetry, control orders, sonar data, derived sensor-based data, etc.), and communication between various levels of the control architecture (execution, tactical, strategic levels) or between multiple autonomous vehicles, software agents, or human controllers. This mission control language is a subject of ongoing research and is likely to change significantly in the next version.

A valid document will have a *missiondata, mission, report* or individual command root element. A *missiondata* element can contain up to one *mission* element and an arbitrary sequence of individual *command, report, telemetry*, *control order* and *sonar* data elements. A *mission* element will contain one or more command elements in any order.

a. *"MissionData" Element*

Autonomous vehicle relevant info: mission commands, control orders, telemetry, sonar results, and/or reports to and from internal or external entities (other vehicles, agents, or human controllers).

S/N	Name	Description	Format	Default	Required
1.	UnitsOfMeasure	Units of measure selections for application scaling as required.	XML element.	-	
2.	vehicleName	Name of vehicle.	VehicleTypes = {"aries", "remus", "phoenix", "Los Angeles SSN", "SDV-9"} Enumerated list of potential vehicle types for use with this schema.	aries	N
3.	date	Date and time of mission.	"dd MMM yyyy hh:mm:ss" format; e.g., "15 January 2004 12:59:59"	-	N

Table 7. XML Elements and attributes of *MissionData* element.

b. *"UnitsOfMeasure"Element*

S/N	Name	Description	Type	Default	Required
1.	distance	Units of measurement for distance.	DistanceMeasures = {"feet", "meters", "kilometers", "miles"} Enumerated list of possible distance measurement units.	meters	N

2.	angle	Units of measurement for angle.	AngleMeasures = {“radians”, “degrees”, “rads”} Enumerated list of possible angular measurement units.	degrees	N

Table 8. XML Elements and attributes of UnitsOfMeasure element.

c. *“Mission” Element*

Mission is an ordered set of command elements comprising a vehicle mission. The list of ARIES AUV-specific Execution-level command elements are given in Appendix B.

```
<?xml version="1.0" encoding="utf-8"?>
<MissionData vehicleName="aries" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="C:\auv\Workbench\Scripts\auvCommandLanguage-
xfsp.xsd">
      <Mission>
            <Position x="12" y="55" depth="5"/>
            <Standoff range="3.0"/>
            <Thrusters on="false"/>
            <Waypoint x="120" y="55" z="15"/>
            <Waypoint x="120" y="65" z="15"/>
            <Hover x="12" y="65" heading="270"/>
            <Thrusters on="false"/>
            <Waypoint x="12" y="55" z="5"/>
            <Waypoint x="120" y="55" z="15"/>
            <Waypoint x="12" y="55" z="5"/>
            <Hover/>
            <Wait time="10"/>
            <Depth value="0"/>
            <Wait time="50"/>
            <Thrusters on="false"/>
            <Quit/>
      </Mission>
</MissionData>
```

Figure 28. A sample XML-based mission script [after Hawkins 2002].

3. Mission Script Authoring Tools

The two-dimensional Mission Planner module provides the means to graphically and intuitively display and author the XML-based AUV mission scripts. The user is presented with a two-dimensional planar view of the mission. This module has a mission canvas whereby a user can easily manipulate positional information pertaining to the script. Extra effort was made to ensure that it is context-sensitive and is at the same time as intuitive as possible. For example, adding or deleting a waypoint is a simple right mouse click, double clicking on a point displays the attributes associated to it. Each mission command has its own set of attributes and its respective user interfaces to manipulate them.

Additionally, mission points can be edited manually or adjusted by using the drag feature. Minor, but useful features such as snapping to the grid display were added too.

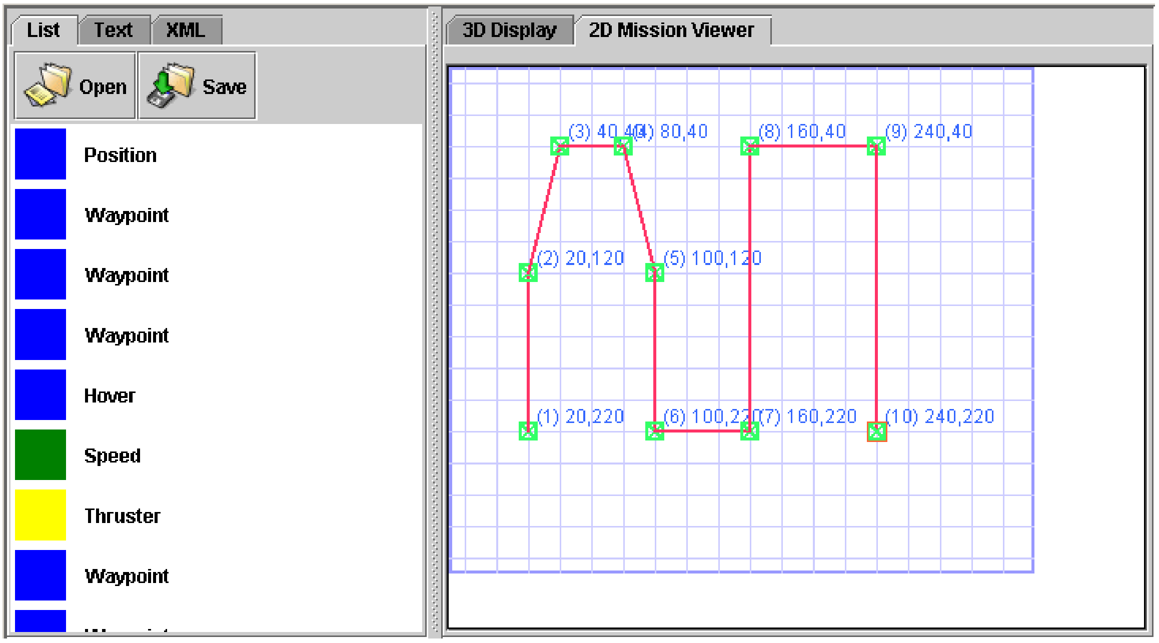


Figure 29. XML-based mission script display and 2D Mission Planner. The mission commands are displayed as a list on the left and the positional data are displayed graphically on the right.

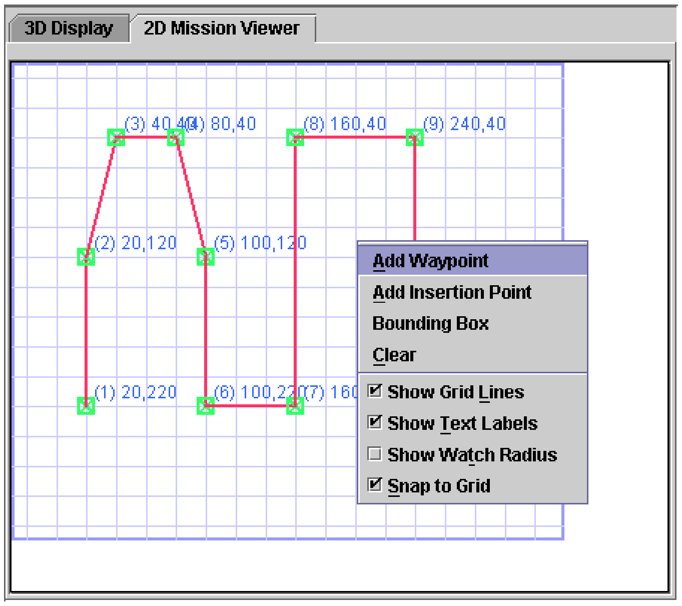


Figure 30. Right-click popup menu for the 2D Mission Planner. The popup menu provides the user with additional functionalities (e.g., add a "Waypoint").

S/N	Name	Description
1.	Add Waypoint	Add a new waypoint.
2.	Add Insertion Point	Add or update start or insertion point.
3.	Bounding Box	Defines a rectangular area of interest
4.	Clear	Clear the mission script.
5.	Show Grid Lines	Display grid lines (25 pixels apart).
6.	Show Text Labels	Display the text labels associated to each point.
7.	Show Watch Radius	Display the watch area circle around each point.
8.	Snap to Grid	Position or align to the grid lines.
9.	Background Color	Set the background color.

Table 9. Details of mouse right-click popup menu items.

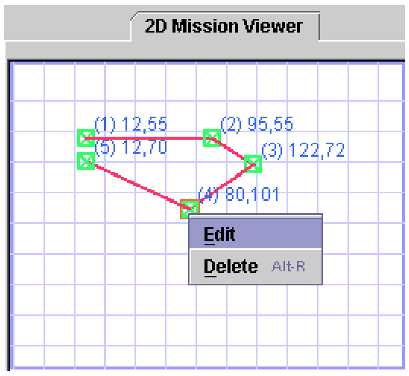
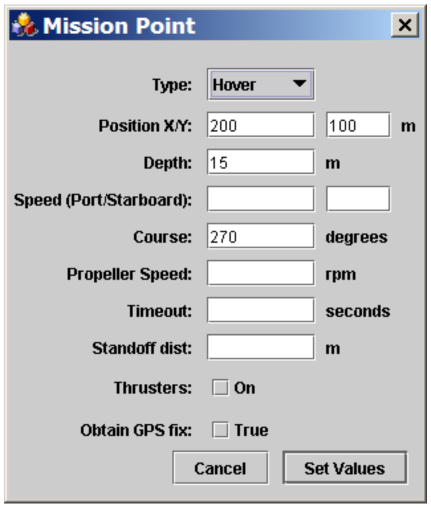
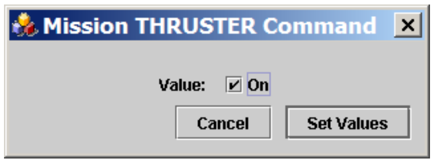


Figure 31. Select a point and right-click to either "Edit" or "Delete" a waypoint.



Type:	Hover		
Position X/Y:	200	100	m
Depth:	15	m	
Speed (Port/Starboard):			
Course:	270	degrees	
Propeller Speed:		rpm	
Timeout:		seconds	
Standoff dist:		m	
Thrusters:	☐ On		
Obtain GPS fix:	☐ True		

Figure 32. Mission Command Editor showing the Waypoint information.



Value: ☑ On

Figure 33. Mission Command Editor showing the Thruster information.

A two-dimensional viewer was developed to facilitate mission generation since it is easier for a user to manipulate in 2D space than 3D space. Depth information can be displayed alongside each 2D point or through the use of color coding. While the AUV mission script has a rich set of commands including non-positional ones, only mission commands that contain positional information are graphically displayed since they contain x-y coordinates. To address this deficiency, an XML-based Mission List module was developed to run alongside the 2D Mission Planner. The Mission List module is essentially a list of all mission commands in the current script. The two graphical views are currently linked dynamically with changes made on either side automatically reflected on the other.

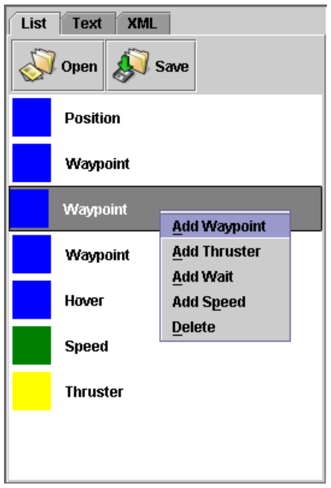


Figure 34. Right-click popup menu on the Mission List display.

D. EXECUTION AND DYNAMICS PROCESSES

1. Execution

The mission execution module uses the same software that is on board the actual AUV. Utilization of the actual AUV software facilitates the development of control equations and algorithms, and enables the realistic rehearsal and fine-tuning of missions in a benign lab environment prior to attempting their execution in open water. By querying the mathematical model of the virtual world for telemetry data rather than onboard sensors, the AUV software can create for itself the illusion that it is operating in the water and the software will behave accordingly.

The AUV Workbench currently has two versions of the AUV execution software. The primary differences are the implementation programming language (compiled C code and Java) and useable command language options. The Java version supports both simple textual and XML-based mission scripts [Ayala 2002], whereas the compiled C version only supports text-based scripts. The former is preferred due to its support for XML-based mission scripts. XML helps to remove any ambiguity in the names of the commands and provides error-checking through validation.

The vehicle behavior can be adapted to other vehicles by adjusting the control constants and by adding, deleting or changing control equations. The control algorithms can be tested and visualized with various mission scripts, against known hydrodynamics models. Any effort to provide precision control for an AUV requires an accurate estimation of both the vehicle's physical and hydrodynamic parameters. Here a vehicle model for controlled steering behaviors was developed and the hydrodynamic parameters were calculated from actual data obtained from operations. [Johnson 2001]

2. Dynamics

The virtual world dynamics thread implements the AUV hydrodynamics mathematical model. When passed a telemetry string from the AUV execution thread, the model is applied, and then a follow-on telemetry string is generated to pass back to the AUV. Additionally, a Distributed Interactive Simulation (DIS) packet is broadcast over the network to drive the visualization thread of the workbench (as well as any other DIS-enabled visualization application that may be on the network). In addition to hydrodynamics modeling, the dynamics thread contains classes that are utilized to model the vehicle's onboard sensors. Sonar data (or that from any other onboard sensor) can therefore be derived and encapsulated within the telemetry string and DIS packets to allow for realistic feedback to the AUV execution software, and accurate mission visualization by the human operator. As with the execution software, the hydrodynamics mathematical model and sensor models currently in use were developed to model the vehicles operated by NPS, but can be arbitrarily adapted to other vehicles simply by modifying the control constants.

The effects of the surrounding environment on a robot vehicle are unique to underwater domain. Understanding these forces is a key requirement in the development and control of the vehicle behavior. The dynamics program Java source code is designed to substitute for the natural environment effects on the AUV. It also provides an estimate of the AUV behavior in the water by performing a series of calculations using physical laws. By communicating with the execution code via a network socket, the telemetry data or state variables of the vehicle are collected. Dynamics apply several equations of motions, forces, and accelerations to the hydrodynamics model and the data received from the execution code. The data produced by dynamics is then sent back to execution, where it is analyzed and appropriate action commands are then given to the respective actuators based on that data. This is a important and difficult part in the real-time simulation in a virtual world [Ayala 2002].

The 3D visualization algorithms in the dynamics code allow the update of 3D scenes developed using X3D-Edit. These scenes are viewed through an Internet browser using a plug-in VRML viewer.

E. 3D VISUALIZATION

1. Design and Implementation

The visualization portion of the workbench contains a 3D viewer that utilized X3D or VRML models of the AUV and its virtual environment. The 3D viewer is developed using an open-source 3D library, Xj3D. By reading and interpreting the incoming DIS packets from network, the viewer automatically animates the vehicle. Through the 3D display, the user is provided with visual feedback on control settings, sensor effectiveness and utilization.

2. User Interface

The 3D display module is located in the upper right pane of the application window. It is on the tabbed page component alongside the 2D mission viewer module.

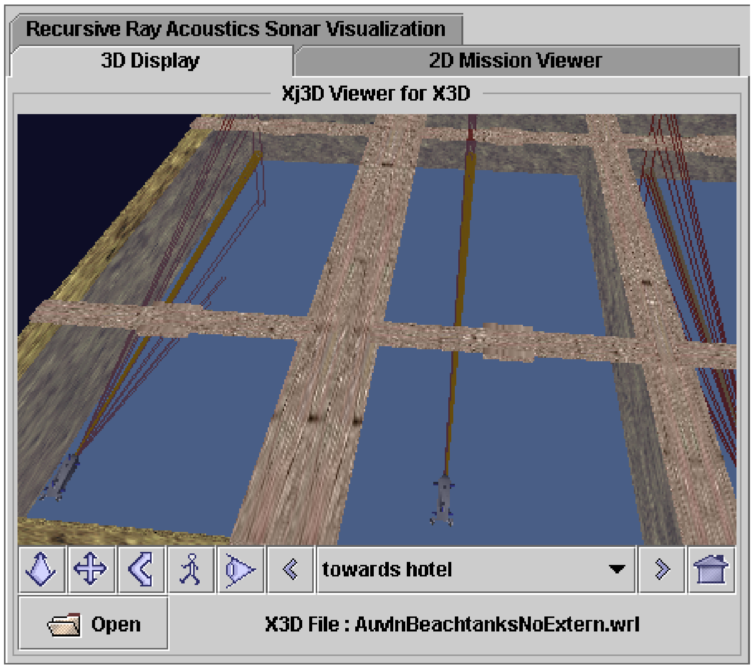

Recursive Ray Acoustics Sonar Visualization
3D Display
2D Mission Viewer
Xj3D Viewer for X3D
towards hotel
Open
X3D File : AuvInBeachtanksNoExtern.wrl

Figure 35. 3D Visualization Display displaying *AUVInBeachTank* scene.

F. WEB SERVER

1. Design and Implementation

In a collaborative environment, there is always a need to share information. For example, data such as current position and list of obstacles encountered can be published and easily accessible to both human operators and other planners who may or may not be using the Workbench. Dissemination of information via web server is well tested and has proven to be a stable and efficient solution. One possible way to web-enable the Workbench is to deploy a full-fledged open-source web server such as Apache Tomcat, but this approach introduces additional deployment, administrative and maintenance issues. Therefore a scaled down but fully functional multi-threaded web server has been incorporated as a module in the AUV Workbench. This allows publishing of information directly from the Workbench. At the same time, the web server is able to process uploaded files via HTTP POST. Through the use of HTTP GET and HTTP POST, it is possible to incorporate message and file sharing capability via HTTP into the Workbench. It has been proposed that the HTTP file transfer mechanism be used for "mirroring" of mission data between individual Workbench applications and a central archival server.

S/N	Directory name	Description
1.	dataweb	Default location.
2.	dataweb/in	Location to store incoming data via HTTP POST.
3.	dataweb/results	Location to store XSBC generated AUV mission telemetry data.

Table 10. Sample web server directory structure.

The web server settings are stored in the XML-based configuration:

```
<Webserver docroot="../defaultroot/" port="80"
           autostart="true" upload="../defaultroot/data/"/>
```

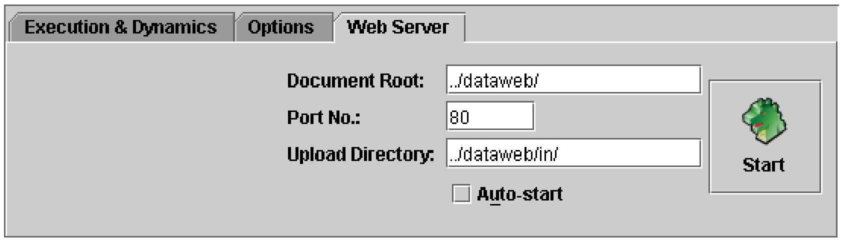
Figure 36. Web server settings in XML configuration file.

S/N	Name	Type	Description
1.	Webserver	Element	Web server parameters.
2.	docroot	Attribute	Web server default directory.
3.	Port	Attribute	Web server port no.
4.	autostart	Attribute	Auto-start web server upon application startup?
5.	Upload	Attribute	Location to store uploaded files.

Table 11. XML tagset defining the web server configuration.

2. User Interface

The web server module is located on the lower pane of the Workbench application, on the *Web Server* tabbed page. A set of default values, such as document default directory and port number, are given. Simply clicking on the *Start* button invokes the web server. To test whether the web server is working, open a hyperlink to the host or machine that the AUV Workbench application is running on, using an Internet browser; e.g., http://localhost:80/index.htm.



Document Root: ../dataweb/
Port No.: 80
Upload Directory: ../dataweb/in/
Auto-start

Figure 37. Web server module user interface.

S/N	Component	Description
1.	Document Root	Web server default directory.
2.	Port No.	Web server port number; e.g., 8080
3.	Upload directory	Location to store uploaded files.
4.	“Auto-start” checkbox	Automatically start the web server upon application start-up?
5.	“Start” button	Start or stop the web server

Table 12. Details of web server user interface.

G. JABBER INSTANT MESSAGING

1. Design and Implementation

To facilitate near-realtime communications, a customized Java-based Jabber client is incorporated into the Workbench. The customized client is able to handle simple plain-text messages and binary file data (e.g., images and XML-based mission script). The *XTC Monitor* module is built on top of the customized Jabber client. An open-source Jabber library, JiveSoftware Smack library is used [JiveSoftware 2003]. Section H covers the XTC Monitor in a greater depth. Worth noting is that the customized Jabber module is introduced for *XTC Event Monitoring* and packaging of a binary file. It is not to replicate the simple human-to-human text messaging capability found in standard Jabber clients.

S/N	Directory name	Description
1.	dataim/in	Directory location to store incoming decoded XHTML binary data.
2.	dataim/out	Directory location to store outgoing binary file data.

Table 13. Workbench instant messaging directory structure.

The Jabber settings are stored in the XML-based configuration:

```
<Jabber dirIn="../dataim/" domain="surfaris.cs.nps.navy.mil"
        port="5222" username="lee" nickname="XJava" resource="Work"
        jid="savage@conference.xchat.movesinstitute.org"/>
```

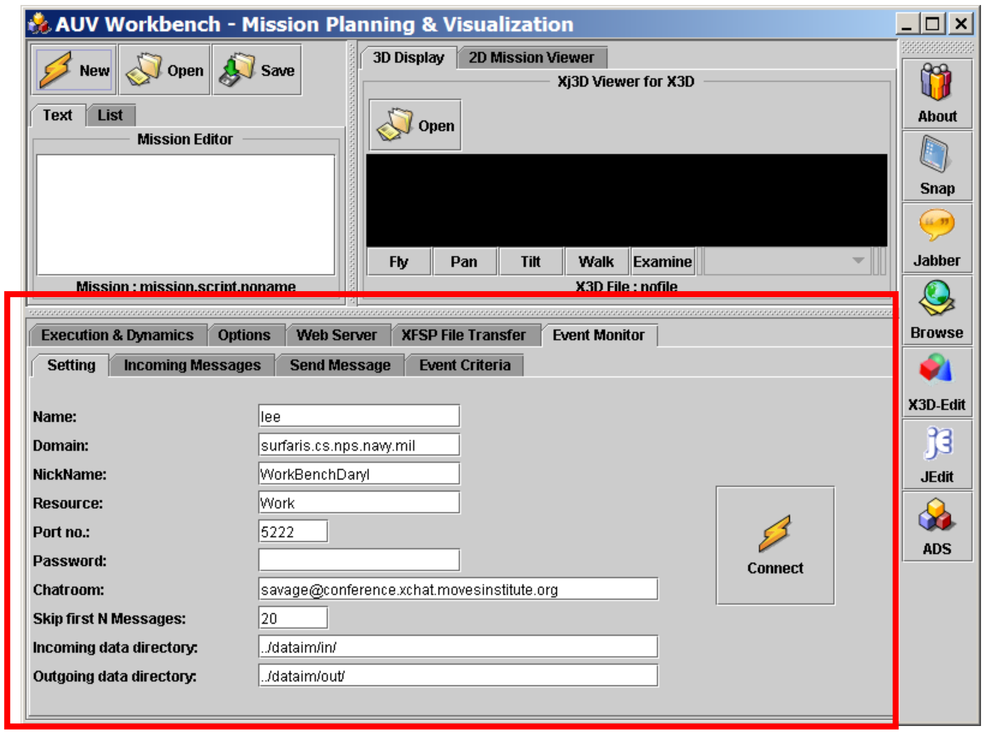
Figure 38. Jabber settings defined in the AUV Workbench configuration file

S/N	Name	Type	Description
1.	Jabber	Element	Web server parameters.
2.	dirIn	Attribute	Directory location to store incoming decoded XHTML binary data.
3.	dirOut	Attribute	Directory location to store outgoing binary file data.
3.	domain	Attribute	Jabber host.
4.	port	Attribute	Jabber port number.
5.	username	Attribute	Login user name
6.	nickname	Attribute	Nickname.
7.	resource	Attribute	Resource.
8.	jid	Attribute	JID or chat-room to listen to upon login.

Table 14. XML tagset specifying the Jabber configurations.

2. User Interface

This module is located on the lower pane of the Workbench application, on the *XTC Monitor* tabbed page. The application reads in the Jabber settings under the *Jabber* stanza of the configuration file (Figure 25). This set of values is populated in the edit-boxes within the *Settings* tabbed page (Figure 39). Once the password is set, clicking on *Connect* button establishes a session with the Jabber server specified in the *domain* edit-box. At the same time, the client will start to listen for messages in the chat room (specified in *Chatroom* edit-box).



Name:	lee
Domain:	surfaris.cs.nps.navy.mil
NickName:	WorkBenchDaryl
Resource:	Work
Port no.:	5222
Password:	
Chatroom:	savage@conference.xchat.movesinstitute.org
Skip first N Messages:	20
Incoming data directory:	../dataim/in/
Outgoing data directory:	../dataim/out/

Figure 39. User interface to configure instant messaging (IM) settings.

S/N	Component	Description
1.	Name	User log on name
2.	Domain	Hostname of Jabber server.
3.	Resource	User profile.
4.	Port no.	Port number to be used.
5.	Password	User log on password.
6.	Chat room	Chat room to listen to upon log on.
7.	Skip first N messages	Upon establishing a session, the Jabber server echos the entire list of messages in that chat room. This setting allows the customized client to skip some of the old messages.
8.	Incoming data directory	Location to store decoded binary data.
9.	Outgoing data directory	Location to store binary data to be packaged and sent out.

Table 15. Details of customized Jabber user interface.

H. XTC EVENT MONITOR

1. Design and Implementation

The *XTC Event Monitor* module is comprised of three sub-modules. *IMSend* and *IMReceive* are the two basic ones used for instant messaging. *IMSend* is responsible for the packaging of binary data and sending it out. It is able to handle single or multiple file attachments. The outgoing message may be addressed to a specific Jabber user (i.e., peer-to-peer) or a chat-room. *IMReceive* listens for posted messages. Again, it is listening to either a particular Jabber user or a chat-room. When there is an incoming message, it parses it and using event monitor criteria defined in the *IMCriteria* sub-module, it generates the appropriate response or alert. *IMReceive* is able to re-generate packaged binary files within a Jabber message (i.e., sent by *IMSend* or similar programs). A sample XHTML message with encoded binary file data is shown in Figure 40. See Chapter IV for details on the design and implementation of the message package module. Next, this section discusses how the incoming events are processed, how alerts are raised and ways messages can be sent.

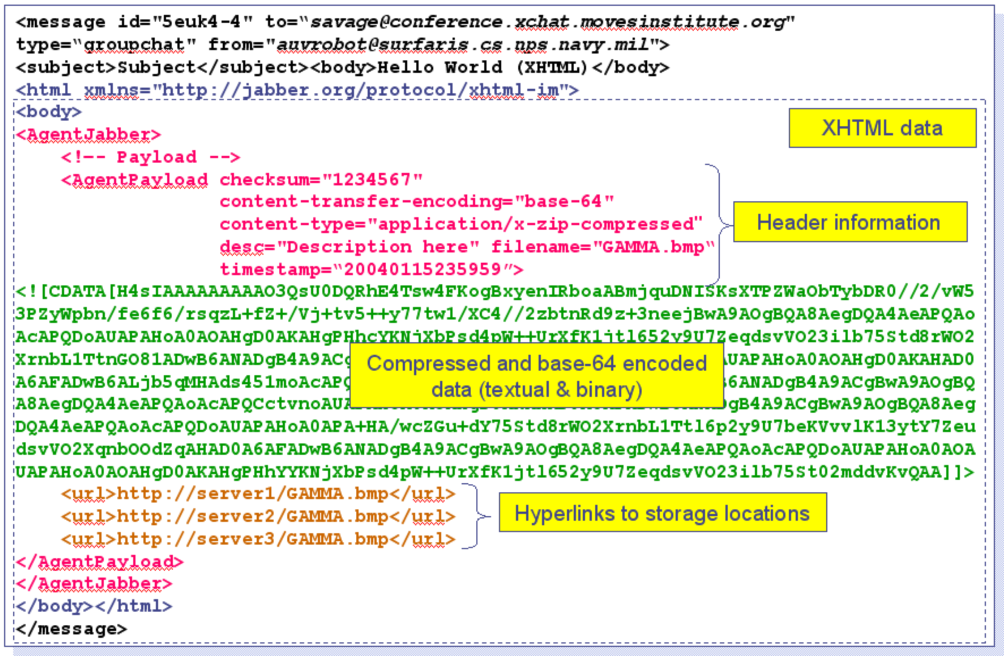
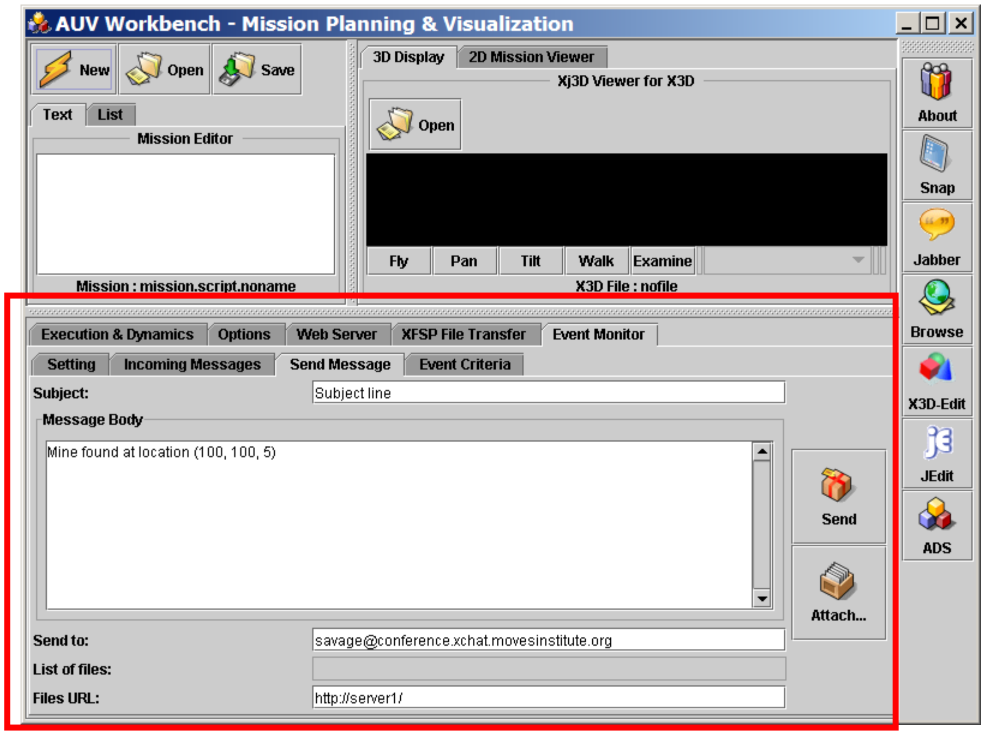

```
<message id="5euk4-4" to="savage@conference.xchat.movesinstitute.org"
type="groupchat" from="auvrobot@surfaris.cs.nps.navy.mil">
<subject>Subject</subject><body>Hello World (XHTML)</body>
<html xmlns="http://jabber.org/protocol/xhtml-im">
<body>
<AgentJabber>
    <!-- Payload -->
    <AgentPayload checksum="1234567"
                  content-transfer-encoding="base-64"
                  content-type="application/x-zip-compressed"
                  desc="Description here" filename="GAMMA.bmp"
                  timestamp="20040115235959">
<![CDATA[H4sIAAAAAAAAAO3QsU0DQRhE4Tsw4FKogBxyenIRboaABmjquDNISKsXTPZWaObTybDR0//2/vW5
3PZyWpbn/fe6f6/rsqzL+fZ+/Vj+tv5++y77tw1/XC4//2zbtnRd9z+3neejBwA9AOgBQA8AegDQA4AeAPQAo
AcAPQDoAUAPAHoA0AOAHgD0AKAHgPHhcYKNjXbPsd4pW++UrXfK1jt1652y9U7ZeqdsvVO23ilb75Std8rWO2
XrnbL1TtnGO81ADwB6ANADgB4A9ACg[illegible]UAPAHoA0AOAHgD0AKAHAD0
A6AFADwB6ALjb5qMHAds451moAcAPQ[illegible]6ANADgB4A9ACgBwA9AOgBQ
A8AegDQA4AeAPQAoAcAPQCctvnoAU[illegible]gB4A9ACgBwA9AOgBQA8Aeg
DQA4AeAPQAoAcAPQDoAUAPAHoA0APA+HA/wcZGu+dY75Std8rWO2XrnbL1Ttl6p2y9U7beKVvvlK13ytY7Zeu
dsvVO2XqnbOOdZqAHAD0A6AFADwB6ANADgB4A9ACgBwA9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0AOA
UAPAHoA0AOAHgD0AKAHgPHhYYKNjXbPsd4pW++UrXfK1jtl652y9U7ZeqdsvVO23ilb75St02mddvKvQAA]]>
    <url>http://server1/GAMMA.bmp</url>
    <url>http://server2/GAMMA.bmp</url>
    <url>http://server3/GAMMA.bmp</url>
</AgentPayload>
</AgentJabber>
</body></html>
</message>
```
XHTML data
Header information
Compressed and base-64 encoded data (textual & binary)
Hyperlinks to storage locations

Figure 40. Sample XHTML message with encoded binary file in *CDATA* section.

The third sub-module, *IMCriteria* is for the definition of event monitoring criteria. For this thesis, regular expressions are used to define the watch events. *Watch Event* determines whether an incoming message matches the regular expression patterns. If a match is found, respective alerts are raised.

2. User Interface

As part of the customized Jabber client, *XTC Monitor* functionalities are found on the same *XTC Monitor* tabbed pages.



Subject: Subject line

Message Body

Mine found at location (100, 100, 5)

Send to: savage@conference.xchat.movesinstitute.org

List of files:

Files URL: http://server1/

Figure 41. Instant Messaging user interface to package and send text and files.

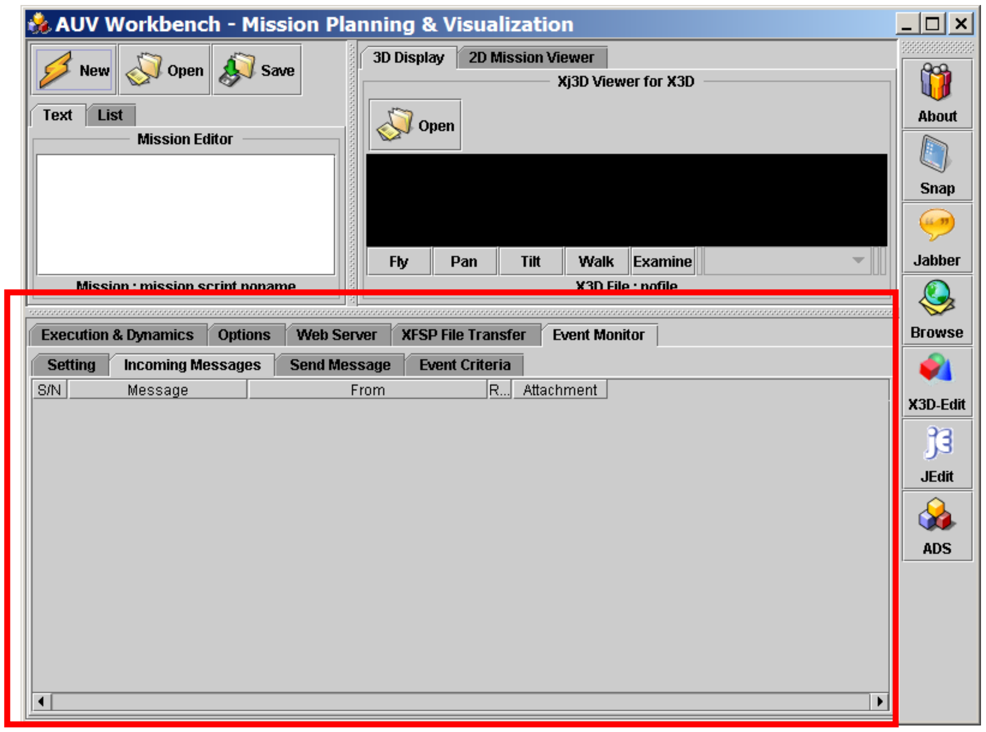


Figure 42. Instant Messaging user interface to display list of incoming messages.

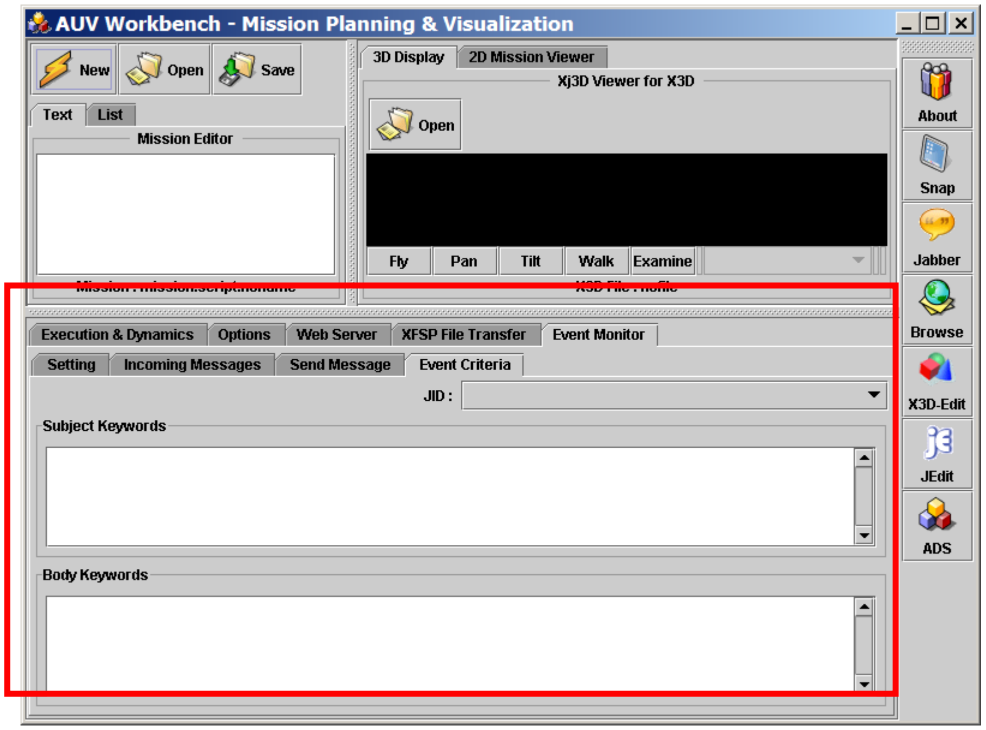


Figure 43. Instant Messaging user interface to define the criteria to alert the user.

3. XTC Event Monitoring Configuration

The settings for the event monitor module are XML-based and included under the *XTCMonitor* stanza of the AUVWorkbench configuration file. There are two types of event monitors: There is one instance of default event monitor (*MonitorDefault*) and multiple instances of Jabber user-specific event monitors (*Monitor*). There can be one or more *Alert* elements associated to a *WatchEvent* element. These alerts, if enabled (i.e., *enabled* attribute set to *true*), will be raised in a consecutive order.

```
<XTCMonitor>
   <MonitorDefault keywordSubject="mine, bomb, torpedo"
keywordBody="nice, mine, bomb, torpedo, location, CVN62, SNN12, DDG51">
      <WatchEvent name="Mine" desc="Look out for Mines"
expr="^.*(?i)MINE[s|S]? .*[ (]{1,2}(\d*),[ ]{0,2}(\d*),[ ]{0,2}(\d*)[
).]?+">
          <Alert type="visual" src="image/mine.gif" enabled="true"/>
          <Alert type="sound" src="sound/alert.wav" enabled="true"/>
          <Alert type="url" src="C:/auv/Workbench/doc/index.htm"
enabled="false"/>
      </WatchEvent>
      <WatchEvent name="Ship" desc="Look out for Ships"
expr="^.*(?i)SHIP[s|S]? .*[ (]{1,2}(\d*),[ ]{0,2}(\d*),[ ]{0,2}(\d*)[
).]?+">
          <Alert type="visual" src="image/ship.gif" enabled="true"/>
      </WatchEvent>
      <WatchEvent name="Location" desc="Look out for Locations"
expr="^.*(?i)LOCATION[s|S]? .*[ (]{1,2}(\d*),[ ]{0,2}(\d*),[
]{0,2}(\d*)[ ).]?+" alert=""/>
   </MonitorDefault>
   <Monitor jid="savage@conference.xchat.movesinstitute.org" desc=""
datetimeStart="" dateTimeEnd="" keywordSubject="mineX, bomb, torpedo"
keywordBody="mineX, bomb, torpedo, location"/>
   <Monitor jid="auvrobot@surfaris.cs.nps.navy.mil" desc=""
datetimeStart="" datetimeEnd="" keywordSubject="urgent, problem"
keywordBody="damage, sinking, surface"/>
</XTCMonitor>
```

Figure 44. Sample *EventMonitor* stanza specifying the type of *Watch Events* and their corresponding *Alerts*.

S/N	Name	Type	Description
1.	EventMonitor	Element	Root element for Event Monitoring stanza.
2.	MonitorDefault	Element	Default event monitor.
3.	WatchEvent	Attribute	Watch event to look for. There can be multiple <WatchEvent> under <MonitorDefault> or <Monitor> elements.
4.	name	Attribute	Name of watch event.
5.	desc	Attribute	Description of watch event.
6.	expr	Attribute	Regular expression to be match against.
7.	Alert	Element	Alert to invoke upon a successful match. There can be multiple <Alert> within a <WatchEvent>.
8.	type	Attribute	Type of alert. An enumeration of “visual”, “sound” and “url”.
9.	src	Attribute	Source of the alert; e.g., image/mine.gif. This will determine how the alert rendered; e.g. if it is a visual one, it is plotted.
10.	Monitor	Element	Event monitor associated to a particular Jabber user ID.
11.	jid	Attribute	Jabber user ID.
12.	desc	Attribute	Description.
13.	datetimeStart	Attribute	When to start this event monitor.
14.	datetimeEnd	Attribute	When to stop this event monitor.

Table 16. XML tagset to configure XTC event monitoring.

4. How Incoming Events are Handled

Upon application startup, the list of default and user-specific event monitors are loaded. For each of the monitors, there can be one or multiple watch events (in *WatchEvent* tag). The check to determine whether a watch event is matched against an incoming event is via the regular expression defined within the *expr* attribute. Once there is a match, the list of available alerts under the *WatchEvent* element is raised. There are three types of alerts: “*visual*”, “*sound*” and “*url*”. The alert type is depicted under the *type* attribute of the *Alert* element. In addition, the *src* attribute defines the source location of the alert; e.g., image or sound path.

```
<WatchEvent expr="^.*(?i)MINE[s|S]? .*[ (]{1,2}(\d*),[ ]{0,2}(\d*),[
]{0,2}(\d*)[ ).]?+">
    <Alert type="visual" src="image/mine.gif"/>
    <Alert type="sound" src="sound/event.wav "/>
</WatchEvent>
```

Figure 45. *WatchEvent* quatrain.

For this thesis, the following alert mechanisms are implemented:

a. Visual Alert

The 2D Mission Planner module handles this alert. A new target object is added to the list of targets, maintained by the module. Next the target is plotted on the 2D display using the image specified in the *src* attribute.

```
<Alert type="visual" src="image/mine.gif"/>
```

Figure 46. A sample alert of type "visual".

b. Sound Alert

This alert is handled within the Event Monitoring module. If the source path of the alert exists, a sound is played back.

```
<Alert type="sound" src="sound/alert.wav"/>
```

Figure 47. A sample alert of type "sound".

c. URL or Hyperlink Alert

The Event Monitoring module opens an application to display the hyperlink or file specified in the *src* attribute. The application is chosen based on the content type of the hyperlink (e.g., .htm) or file (e.g., .bmp or .txt). For example with the following alert (Figure 48), it is of type "url" and the source file ("*C:/auv/Workbench/doc/index.htm*") is of "text/html" content type (determined from the file extension). Based on the "text/html" content type, the module then looks for the associated application to render it.

```
<Alert type="url" src="C:/auv/Workbench/doc/index.htm"/>
```

Figure 48. A sample alert of type "url".

The list of available applications is defined in the AUV Workbench configuration file under the *Application* stanzas. A sample of the *Application* stanza is given in Figure 49.

```
<Application name="Browse" tooltip="Web Browser"
     image="image/browser.gif" show="true"
     content-type="text/html">
   <Command>C:/Program Files/mozilla.org/Mozilla/mozilla.exe</Command>
   <Command>C:/Program Files/Internet Explorer/IEXPLORE.EXE</Command>
</ Application >

<Application name="NotePad" tooltip="Windows Notepad"
     image="image/note.gif" show="false"
     content-type="text/plain">
   <Command>C:/windows/NOTEPAD.EXE</Command>
</Application>

<Application name="Picture" tooltip="Windows Fax and Viewer"
     image="image/graphics.gif" show="false"
     content-type="image/bmp, image/gif">
   <Command>C:/windows/System32/mspaint.exe</Command>
</Application>
```

Figure 49. A sample list of applications defined in *Application* stanzas that can be invoked.

An overview of the process of event monitoring of instant messages and the triggering of alerts is given in Figure 50.

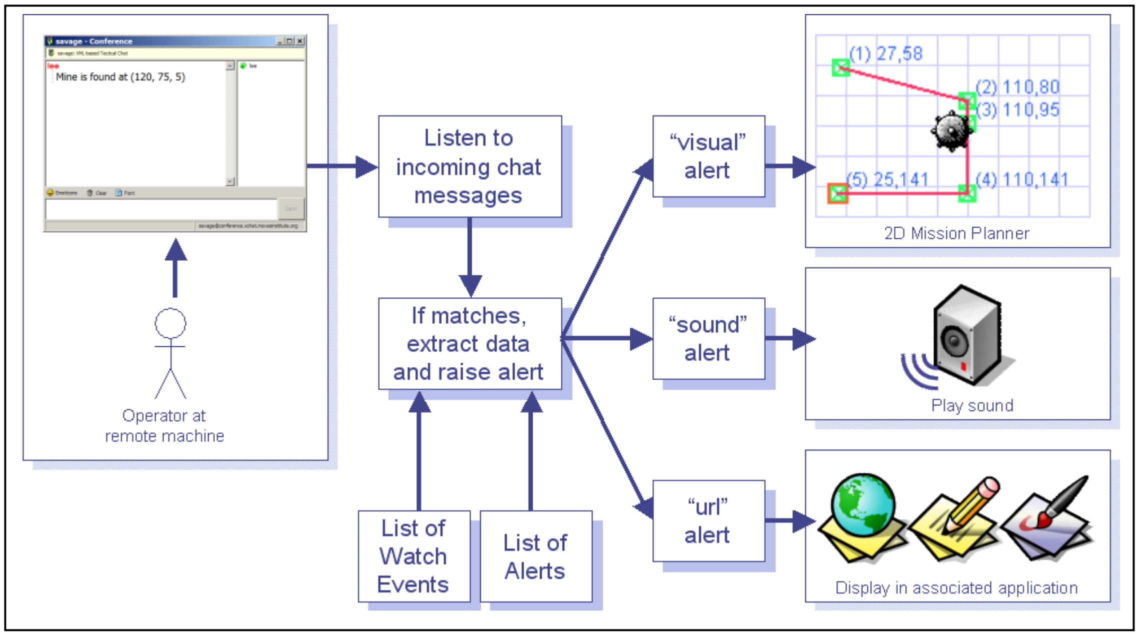


Figure 50. Instant messaging event monitoring and alert mechanism process via standard Jabber client.

5. How Events/Messages are Generated

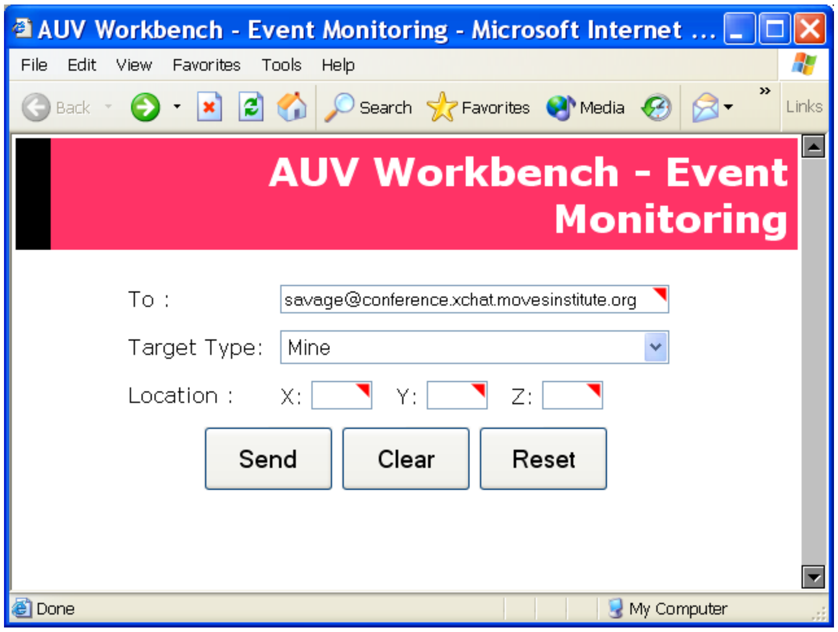
a. Free-form Text Using Standard Jabber Clients

Using a standard Jabber client such as Rhymbox, a human operator keys in the message "*A mine is found at position (100, 100, 5)*" and sends it. With free-form text, the receiving party (i.e., *XTC Monitor*) needs to extract the necessary pieces of information from the chatter. For this thesis, a simple sentence parsing module using Java's Regular Expression Parser was implemented. A more precise and robust extraction module that utilizes Natural Language Processing can be developed in the future. This technique is error-prone especially when the operator is under stress. This in turn leads to additional verification at the software end to ensure that it is tolerable to minor errors such as missing spaces, characters or misplaced characters.

b. Structured Text

Message generation using a structured format is a preferred approach. The user is presented with a form (HTML or Java Swing application). This approach removes the need for a sentence parser and error checking, thus reducing message processing time. Data from structured text comes formatted and possibly validated at the server or client end. The receiving party only needs to extract the necessary portions of data based on a predefined schema. There are various ways that structured text can be captured:

- Capturing the data in the correct format manually by human operators. This is a tedious and error-prine process.
- Use of a customized client application to allow capture and validation of inputs from the operators.
- Use of a web page to allow the operators to key-in the required information. This method is preferred, as it only requires that there is access to a web browser and the web page since this will work as long as the human operator has access to the web page. This removes the need to deploy customized applications.


AUV Workbench - Event Monitoring - Microsoft Internet ...
File Edit View Favorites Tools Help
Back Search Favorites Media Links
Done
My Computer

AUV Workbench - Event Monitoring

To :	savage@conference.xchat.movesinstitute.org
Target Type:	Mine
Location :	X: Y: Z:

Send Clear Reset

Figure 51. An event monitoring HTML form to capture target type and location information.

Advantage of using Instant Messaging protocol is that it allows both human operators and agents to interact in the same environment. Agent-specific data is stored in the XHTML sections of the message. These are not visible on normal Jabber clients, but are caught by agents that are listening for them and processed accordingly.

I. APPLICATION TOOLBAR

The application toolbar is highly configurable to allow users and developers alike to add new applications to the AUV Workbench. The toolbar is both floatable (Figure 52) to increase display real estate on the Workbench as well as dock-able to any side of the Workbench application. This provides a convenient way for users or developers to bundle frequently used applications. The capability is achieved through the use of an XML-based configuration file. The current version of the Workbench has two built-in features: the "*About*" dialog and the "*Screen Capture*" capability (i.e., the first two toolbar buttons). A sample application toolbar is seen below:

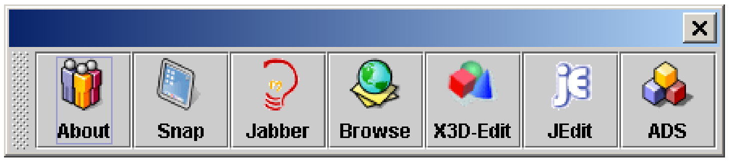


Figure 52. A floating application toolbar.

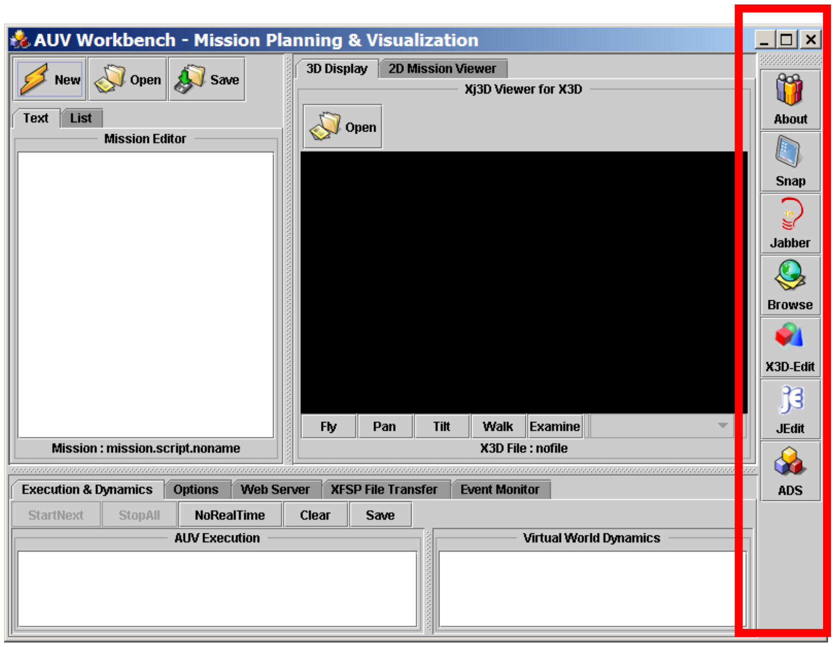


Figure 53. A docked application toolbar on the left.

S/N	Item	Type	Description
1.	About	Built-in	"About" dialog.
2.	Screen-shot	Built-in	Perform a screen capture.
3.	Jabber	External	Standard Jabber client.
4.	Browse	External	Internet browser.
5.	X3D-Edit	External	X3D Graphics Editor.
6.	jEdit	External	Open-source Java Editor.
7.	ADS	External	AUV Data Server.

Table 17. Details of Toolbar Buttons.

The XML-based configuration file is human-readable and nicely partitioned to allow the user to add or remove applications easily. The procedures to add a new application are given below:

- Locate and open the configuration file; e.g., *AUVWorkbenchConfig.xml*.
- Go to the section where the *Application* tags are defined.
- Make a copy of an existing *Application* set.

- Make the necessary changes to the attributes; e.g., `name` is the name that appears in the toolbar button, *tooltip* is the hint and *image* defines the location of the toolbar button image.
- Add the file types that this application can handle under content-type attribute. Set *show* attribute to "*true*" to display in toolbar.
- The *param* attribute defines the parameter to be passed into the application upon its startup; e.g., a hyperlink to be a web-page for a Internet browser application.
- Next add *Command* elements. These define the actual locations of the application. Upon clicking on the particular toolbar button, the AUV Workbench tries to look for the application from the possible list of *Command* tags provided. Once found, the application is invoked.

```
<Application name="Browse" tooltip="Web Browser"
image="image/browser.gif" show="true" content-type="text/html"
param="intranet.nps.navy.mil">
   <Command>C:/Program Files/mozilla.org/Mozilla/mozilla.exe</Command>
   <Command>C:/Program Files/Internet Explorer/IEXPLORE.EXE</Command>
</Application>
```

Figure 54. A sample toolbar application defined in the *Application* stanza.

S/N	Name	Type	Description
1.	Application	Element	Application to be invoked from toolbar.
2.	name	Attribute	Name of application.
3.	tooltip	Attribute	Button tooltip on toolbar.
4.	show	Attribute	Boolean value (true or false). To display in the toolbar or not?
5.	content-type	Attribute	File types that this application can handle. Delimited by commas; e.g., image/bmp, image/gif
6.	param	Attribute	Parameter to be passed into application upon invocation; e.g., a hyperlink to a web-page.
7.	Image	Attribute	Location of image icon on toolbar.
8.	Command	Element	Command to invoke the application. There can be multiple <Command> associated to an application.

Table 18. XML tagset for configuring the toolbar module.

J. STORAGE, NETWORKING AND COMPRESSION

1. Naming Convention

Code conventions are important to programmers since 80% of the lifetime cost of a piece of software goes to maintenance. Software is hardly maintained for its whole life by the original author. Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

S/N	File Type	Suffix
1.	Java Source Code	.java
2.	Java Bytecode	.class
3.	VRML	.vrml
4.	X3D	.x3d

Table 19. File types and their suffixes.

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can be helpful in understanding the code.

S/N	Identifier Type	Rules for Naming	Examples
1.	Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	org.w3c.dom.* javax.xml.parsers.*

2.	Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class WatchEvent;
			class Monitor;

3.	Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate;
			interface Storing;

4.	Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word	run();
			runApp();
			getStatus();

S/N	Identifier Type	Rules for Naming	Examples
		capitalized.	

5.	Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.
		
		Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.
			Int i;
			char c;
			float iSpeed;

6.	Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int FRM_WIDTH = 400;
			
			static final int MAX_WIDTH = 999;
			
			static final String HTTP_Accept = "Accept:"

Table 20. Java Source Code Naming Convention [after JavaCodeConvention 1999].

S/N	Item
1.	CamelCaseNaming: capitalize each word, never use abbreviations, strive for clarity, and be brief but complete.
2.	Ensure consistent capitalization throughout. Of note: Windows systems are not case sensitive, but http servers are. Thus mismatched capitalization can hide target files, and this error only is revealed when placed on a server.
3.	Naming conventions apply to .x3d files, image files, and Prototypes. It is also a good idea to follow them for DEF/USE names.
4.	startWithLowerCaseLetter when defining field names for Prototypes and Scripts. This approach matches the node and field naming conventions in the X3D Specification. When multiple files pertain to a single entity, start with the same name so that they will alphabetize adjacent to each other in the catalog and the directory listings. Examples: WaypointInterpolatorPrototype.x3d WaypointInterpolatorExample.x3d WaypointInterpolatorExample.png
5.	Good choice of directory & subdirectory names can help keep scene names terse.

Figure 55. X3D Naming Convention [X3DHints 2004].

K. TOOLS AND PRODUCTS

1. Overview

This section provides a brief description on the applications that are bundled with the current version of AUV Workbench. There are two built-in functionalities in the Workbench's application toolbar, namely the *About* and *Screenshot* buttons. As the name implies, the *About* button pops up an image that describes the AUV Workbench. The image can be easily changed by replacing the image named *SplashScreen.jpg* in the */image* subdirectory. As a collaboration tool, there is always a need to share the current picture in the AUV Workbench. This may include a view of the mission script and the 3D view of the environment. Thus a fast one-button click screen-capture capability has been added to the Workbench.

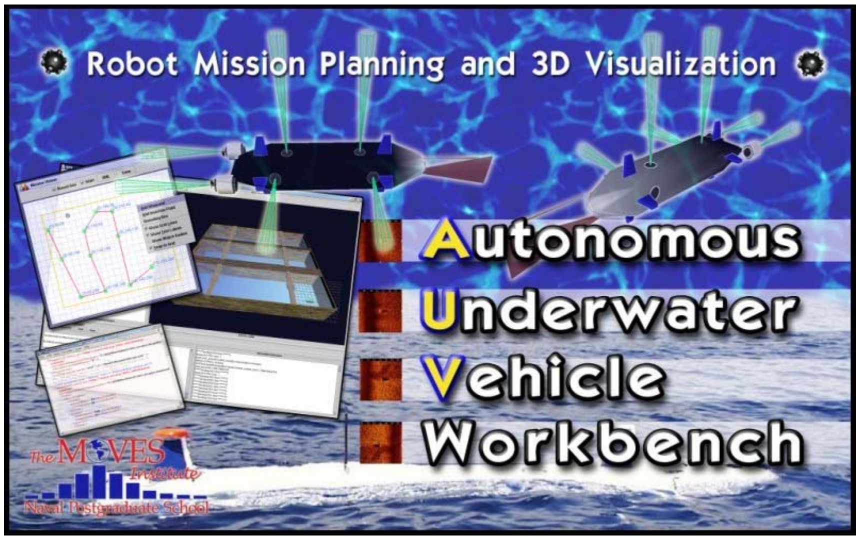

Robot Mission Planning and 3D Visualization
Autonomous
Underwater
Vehicle
Workbench
The MOVES Institute
Naval Postgraduate School

Figure 56. Splash-screen poster image describing the AUV Workbench, produced by the author.

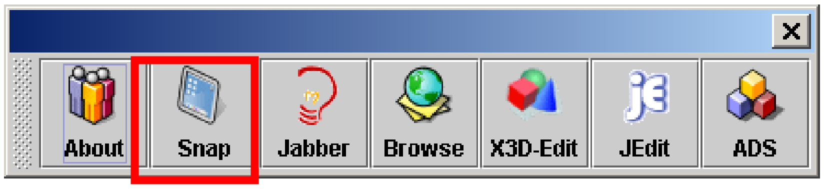


Figure 57. Screen-capture button on the Application Toolbar.

2. Jabber Instant Messaging (IM) Client

A useful tool to facilitate near-real time text messaging between human operators is an Instant Messaging (IM) client. Through the Jabber client, a developer can log onto a AUV Workbench chat-room and post questions or answers to fellow developers. Similarly, human operators are able to use the Jabber client to post mission related information (e.g., location of a mine). All messages posted via the Jabber clients can be logged on the Jabber server. This is useful for post-mission analysis by operational users or consolidation of a trouble-shooting guide for developers.

```
<Application name="Jabber"
    tooltip="Instant Messaging Client" image="image/jabber.gif">
   <Command>C:/Program Files/RhymBox/RhymBox.exe</Command>
   <Command>D:/Program Files/RhymBox/RhymBox.exe</Command>
   <Command>C:/Program Files/IM/RhymBox/RhymBox.exe</Command>
</Application>
```

Figure 58. Jabber application setting in the AUV Workbench configuration file

RhymBox is a Jabber client for instant messaging. The Jabber network employs a distributed and secure infrastructure. The Jabber protocol is based on the IETF supported XMPP. Jabber is also linked to legacy services (e.g., Yahoo!, MSN, AIM, ICQ, etc).

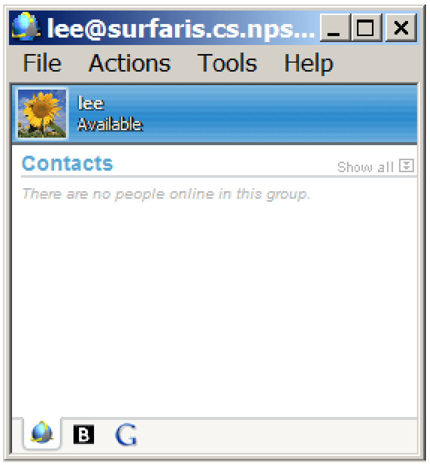
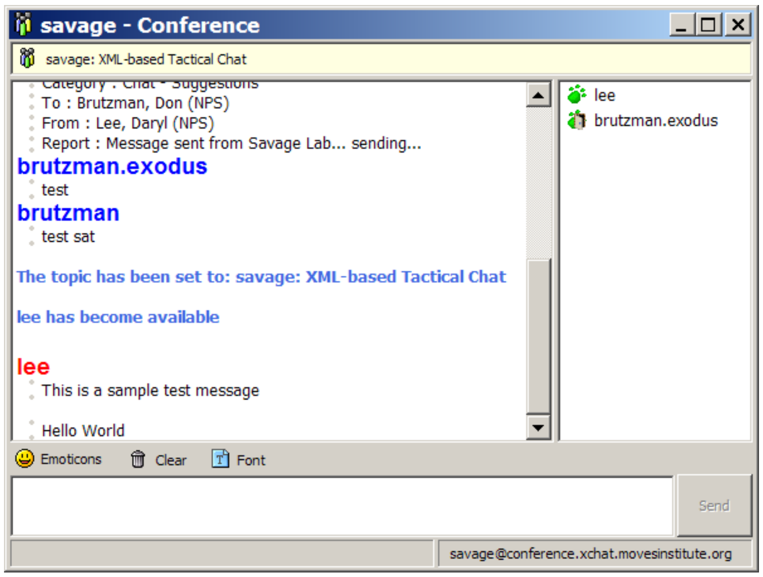


Figure 59. Rhymbox Jabber client main user interface.


savage - Conference
savage: XML-based Tactical Chat
Emoticons
Clear
Font
Send
savage@conference.xchat.movesinstitute.org
lee
brutzman.exodus

Category : Chat - Suggestions
To : Brutzman, Don (NPS)
From : Lee, Daryl (NPS)
Report : Message sent from Savage Lab... sending...

brutzman.exodus
test

brutzman
test sat

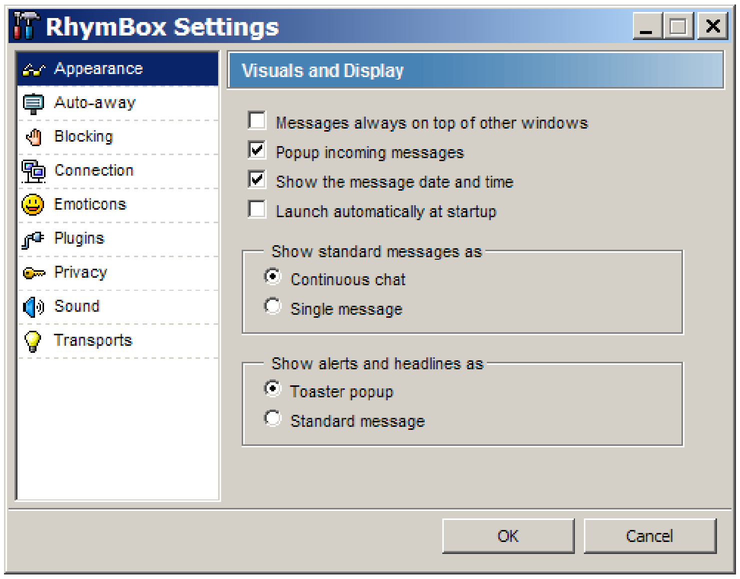
The topic has been set to: savage: XML-based Tactical Chat

lee has become available

lee
This is a sample test message

Hello World

Figure 60. Rhymbox Jabber client "Chat-room" interface.

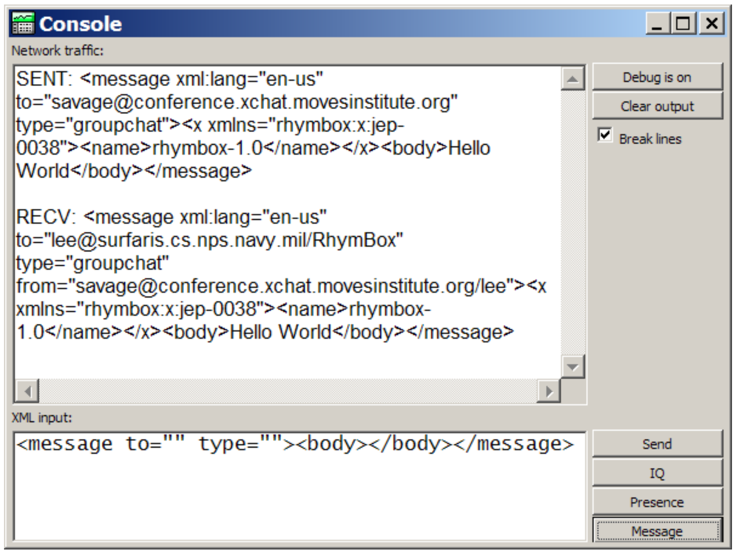


[ ] Messages always on top of other windows
[✓] Popup incoming messages
[✓] Show the message date and time
[ ] Launch automatically at startup

Show standard messages as
[● Continuous chat] Single message

Show alerts and headlines as
[● Toaster popup] Standard message

Figure 61. Rhymbox Jabber client "Settings" interface.



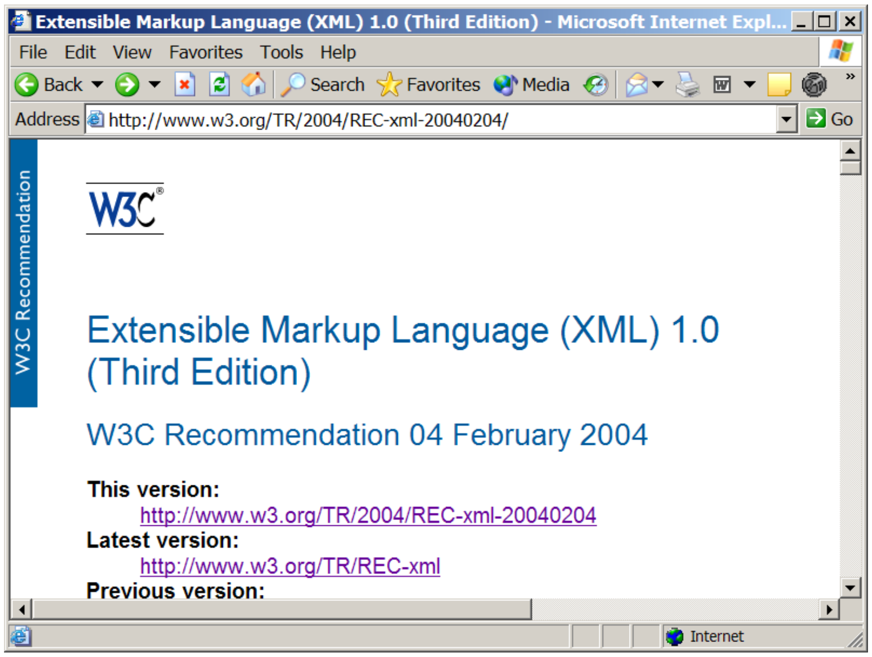
```
SENT: <message xml:lang="en-us"
to="savage@conference.xchat.movesinstitute.org"
type="groupchat"><x xmlns="rhymbox:x:jep-
0038"><name>rhymbox-1.0</name></x><body>Hello
World</body></message>

RECV: <message xml:lang="en-us"
to="lee@surfaris.cs.nps.navy.mil/RhymBox"
type="groupchat"
from="savage@conference.xchat.movesinstitute.org/lee"><x
xmlns="rhymbox:x:jep-0038"><name>rhymbox-
1.0</name></x><body>Hello World</body></message>
```

```
<message to="" type=""><body></body></message>
```

Figure 62. Rhymbox Jabber client "Console" interface.

3. Internet Browser

One of the essential tools required in a collaboration environment is an Internet Browser. The browser allows both users and developers to access the World Wide Web to find information.


Extensible Markup Language (XML) 1.0 (Third Edition) - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.w3.org/TR/2004/REC-xml-20040204/
Go
W3C Recommendation
Internet

Extensible Markup Language (XML) 1.0 (Third Edition)

W3C Recommendation 04 February 2004

This version:
http://www.w3.org/TR/2004/REC-xml-20040204
Latest version:
http://www.w3.org/TR/REC-xml
Previous version:

Figure 63. Microsoft Internet Explorer 6.0 browser user interface.

Once the "Browser" button is clicked, the application looks for the first available browser through the list of possible browser applications, i.e., defined in the *Command* stanza. A default web-page can be specified using the *param* attribute. This allows users to be directed to the relevant web-page (e.g., AUV Workbench development) upon browser startup.

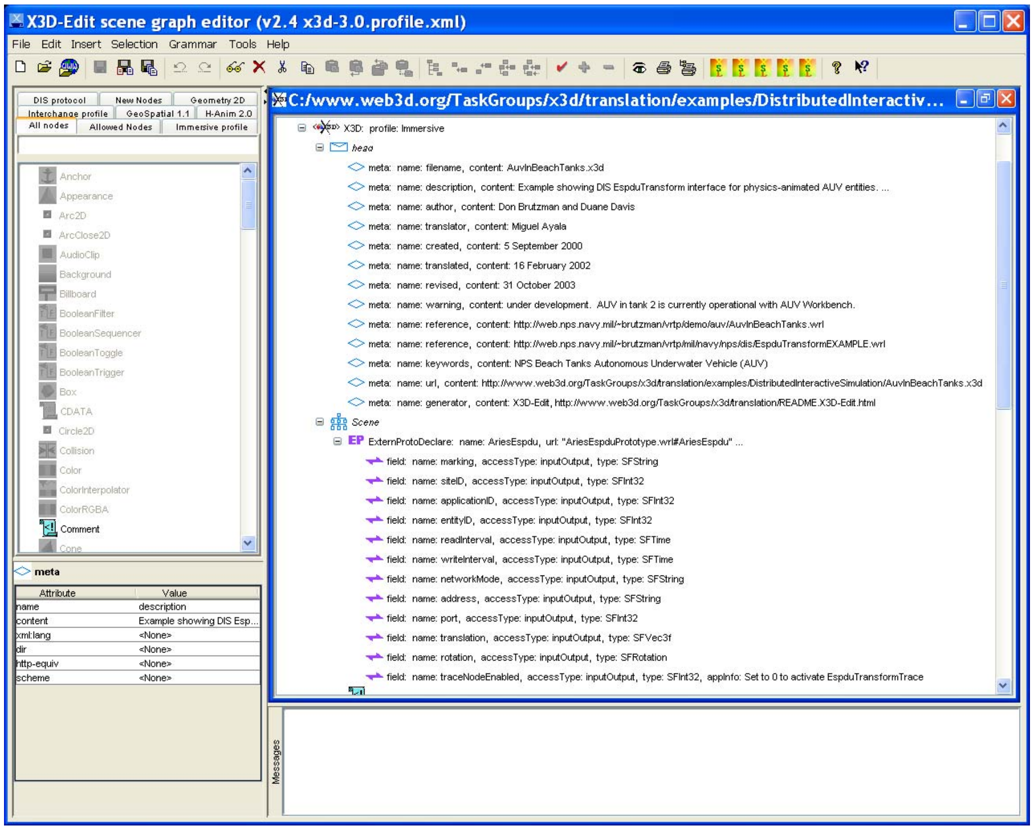
```
<Application name="Browse" tooltip="Web Browser"
             image="image/browser.gif"
             show="true" content-type="text/html"
             param="intranet.nps.navy.mil">
   <Command>C:/Program Files/mozilla.org/Mozilla/mozilla.exe</Command>
   <Command>D:/Program Files/mozilla.org/Mozilla/mozilla.exe</Command>
   <Command>C:/Program Files/Internet Explorer/IEXPLORE.EXE</Command>
   <Command>D:/Program Files/Internet Explorer/IEXPLORE.EXE</Command>
</Application>
```

Figure 64. Internet Browser entry in the AUV Workbench configuration file.

4. X3D-Edit

X3D-Edit is a graphics file editor for Extensible 3D (X3D) Graphics that enables simple error-free editing, authoring and validation of X3D or VRML scene-graph files. Context-sensitive tooltips provide concise summaries of each VRML node and attribute. These tooltips simplify authoring and improve understanding for novice and expert users alike.

X3D-Edit uses the X3D 3.0 tagset defined by the X3D 3.0 Document Type Definition (DTD) in combination with Sun's Java, IBM's Xeena XML editor, and editor profile configuration files. More information on X3D-Edit can be found at http://www.web3d.org/x3d/content/README.X3D-Edit.html.



Attribute	Value
name	description
content	Example showing DIS Esp...
xml:lang	<None>
dir	<None>
http-equiv	<None>
scheme	<None>

```
X3D:  profile: Immersive
  head
    meta:  name: filename,  content: AuvInBeachTanks.x3d
    meta:  name: description,  content: Example showing DIS EspduTransform interface for physics-animated AUV entities. ...
    meta:  name: author,  content: Don Brutzman and Duane Davis
    meta:  name: translator,  content: Miguel Ayala
    meta:  name: created,  content: 5 September 2000
    meta:  name: translated,  content: 16 February 2002
    meta:  name: revised,  content: 31 October 2003
    meta:  name: warning,  content: under development.  AUV in tank 2 is currently operational with AUV Workbench.
    meta:  name: reference,  content: http://web.nps.navy.mil/~brutzman/vrtp/demo/auv/AuvInBeachTanks.wrl
    meta:  name: reference,  content: http://web.nps.navy.mil/~brutzman/vrtp/mil/navy/nps/dis/EspduTransformEXAMPLE.wrl
    meta:  name: keywords,  content: NPS Beach Tanks Autonomous Underwater Vehicle (AUV)
    meta:  name: url,  content: http://www.web3d.org/TaskGroups/x3d/translation/examples/DistributedInteractiveSimulation/AuvInBeachTanks.x3d
    meta:  name: generator,  content: X3D-Edit, http://www.web3d.org/TaskGroups/x3d/translation/README.X3D-Edit.html
  Scene
    EP ExternProtoDeclare:  name: AriesEspdu,  url: "AriesEspduPrototype.wrl#AriesEspdu" ...
      field:  name: marking,  accessType: inputOutput,  type: SFString
      field:  name: siteID,  accessType: inputOutput,  type: SFInt32
      field:  name: applicationID,  accessType: inputOutput,  type: SFInt32
      field:  name: entityID,  accessType: inputOutput,  type: SFInt32
      field:  name: readInterval,  accessType: inputOutput,  type: SFTime
      field:  name: writeInterval,  accessType: inputOutput,  type: SFTime
      field:  name: networkMode,  accessType: inputOutput,  type: SFString
      field:  name: address,  accessType: inputOutput,  type: SFString
      field:  name: port,  accessType: inputOutput,  type: SFInt32
      field:  name: translation,  accessType: inputOutput,  type: SFVec3f
      field:  name: rotation,  accessType: inputOutput,  type: SFRotation
      field:  name: traceNodeEnabled,  accessType: inputOutput,  type: SFInt32,  appInfo: Set to 0 to activate EspduTransformTrace
```

Figure 65. X3D-Edit Graphical User Interface (GUI) for developing 3D objects and scenes using X3D.

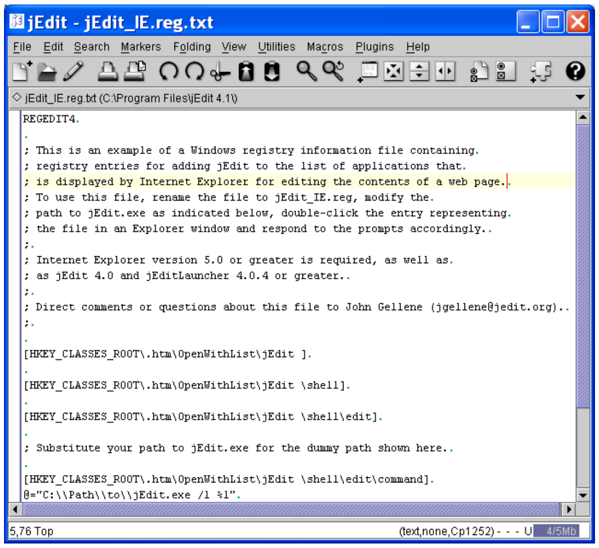
```
<Application name="X3D-Edit"
             tooltip="X3D Editor" image="image/x3d.gif">
   <Command>C:/www.web3d.org/TaskGroups/x3d/translation/X3D-Edit.bat
   </Command>
   <Command>D:/www.web3d.org/TaskGroups/x3d/translation/X3D-Edit.bat
   </Command>
</Application>
```

Figure 66. X3D-Edit entry in the AUV Workbench configuration file.

5. jEdit

jEdit is a cross-platform programmer's text editor written in Java, being developed by Slava Pestov and others. It is available online at http://www.jedit.org. It has an easy to use interface that resembles that of many other Windows and MacOS text editors. It is also highly customizable, and contains a “plugin” architecture that allows its features to be extended by additional programs.

jEdit contains a large assortment of features for manipulating source code, markup text, and other text files. As a programmer's text editor, it also has many features to help programmers manage their projects and work with other programming tools.

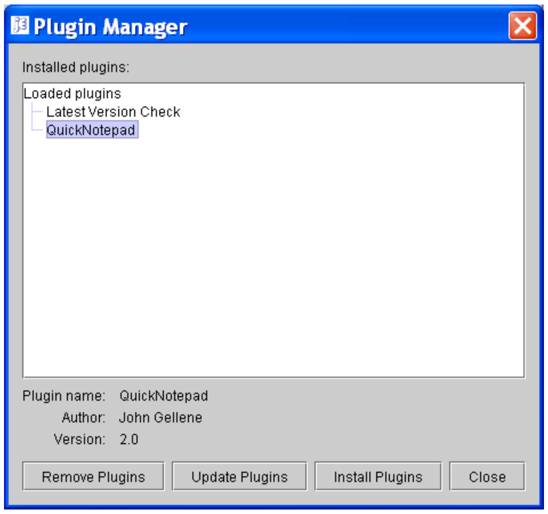

jEdit - jEdit_IE.reg.txt
File Edit Search Markers Folding View Utilities Macros Plugins Help
jEdit_IE.reg.txt (C:\Program Files\jEdit 4.1\)
5,76 Top
(text,none,Cp1252) - - - U 4/5Mb

```
REGEDIT4.
.
; This is an example of a Windows registry information file containing.
; registry entries for adding jEdit to the list of applications that.
; is displayed by Internet Explorer for editing the contents of a web page..
; To use this file, rename the file to jEdit_IE.reg, modify the.
; path to jEdit.exe as indicated below, double-click the entry representing.
; the file in an Explorer window and respond to the prompts accordingly..
;.
; Internet Explorer version 5.0 or greater is required, as well as.
; as jEdit 4.0 and jEditLauncher 4.0.4 or greater..
;.
; Direct comments or questions about this file to John Gellene (jgellene@jedit.org)..
;.
.
[HKEY_CLASSES_ROOT\.htm\OpenWithList\jEdit ].
.
[HKEY_CLASSES_ROOT\.htm\OpenWithList\jEdit \shell].
.
[HKEY_CLASSES_ROOT\.htm\OpenWithList\jEdit \shell\edit].
.
; Substitute your path to jEdit.exe for the dummy path shown here..
.
[HKEY_CLASSES_ROOT\.htm\OpenWithList\jEdit \shell\edit\command].
@="C:\\Path\\to\\jEdit.exe /l %1".
```

Figure 67. jEdit User Interface running on Windows platform.

Text editing can be different on different operating systems (Carriage Return versus Carriage Return-Line Feed differences), and also some default text editors are notoriously poor (e.g. Windows Notepad), jEdit is bundled with the NPS AUV Workbench. This tool ensures that users can perform simple editing tasks on configuration and output files, thus simplifying use and remote debugging support.

A plugin is an application that is designed to work with jEdit by providing additional features that can be used from within the main program. Often the plugin will provide a visible user interface in a window that can be docked to jEdit's main view window. There are currently over 60 publicly available plugins that provide such services as a Java source code browser, a command-line shell, templated text insertion, and source code project management. They can be downloaded, installed, and kept current from within jEdit's “Plugin Manager”.



Plugin name: QuickNotepad
Author: John Gellene
Version: 2.0

Figure 68. jEdit Plugin Manager.

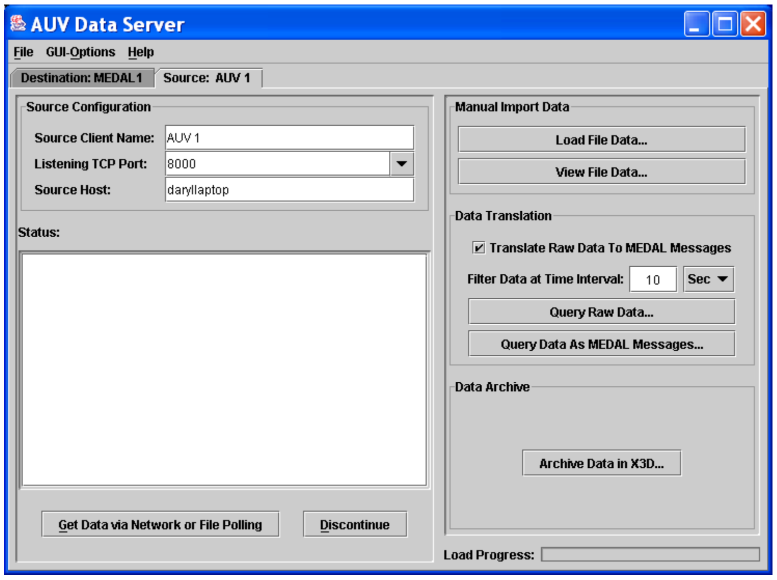
```
<Application name="JEdit" tooltip="JEdit" image="image/jedit.gif">
   <Command>C:/Program Files/jEdit 4.1/jedit.exe</Command>
   <Command>D:/Program Files/jEdit 4.1/jedit.exe</Command>
</Application>
```

Figure 69. jEdit entry in the AUV Workbench configuration file.

The jEdit homepage, located at http://www.jedit.org (Accessed on 28 January 2004) contains the latest version of jEdit, along with plugin downloads. There is also a user-oriented site, http://community.jedit.org (Accessed on 28 January 2004).

6. AUV Data Server

The NPS AUV Data Server (ADS) is a tool for post-mission analysis. It is able to read the actual AUV telemetry data from several different AUV sources and generate a 3D view of the mission in VRML or X3D. ADS has processed data retrieved from the Woods Hole Oceanographic Institution (WHOI), REMUS, Florida Atlantic University Ocean Explorer (OEX), and NPS ARIES AUVs. ADS parses robot telemetry as well as mission asset (track), bathymetry and contact reports. These data files are converted into Message Transfer Format (MTF) message format and imported into Mine Warfare Environmental Decision Aids Library (MEDAL). The MEDAL format is used by the US Navy to evaluate asset positions, mine-like contacts, snipped images of those contacts identified as mines and bathymetry maps. It provides a network message interface to GCCS MEDAL systems and also produces X3D mission visualizations.



Source Configuration

Source Client Name: AUV 1

Listening TCP Port: 8000

Source Host: daryllaptop

Status:

Manual Import Data

Data Translation

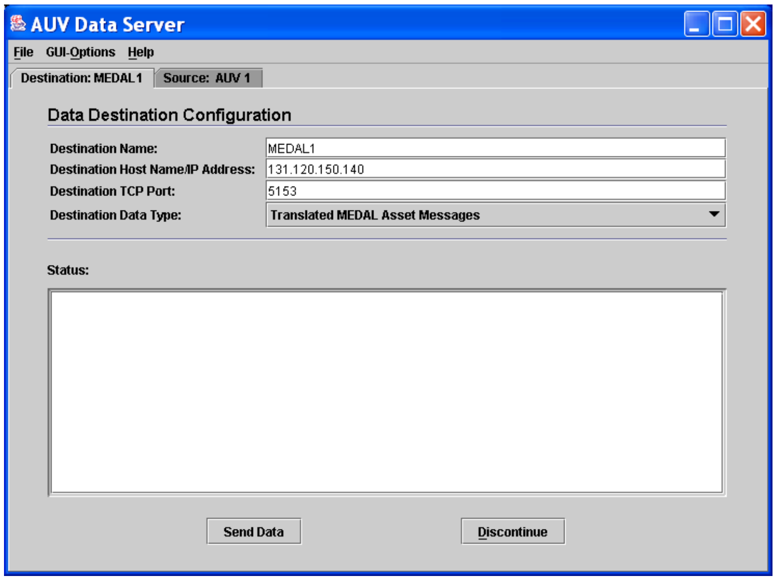
☑ Translate Raw Data To MEDAL Messages

Filter Data at Time Interval: 10 Sec

Data Archive

Load Progress:

Figure 70. *ADS* data source panel user interface.



Data Destination Configuration

Destination Name: MEDAL1

Destination Host Name/IP Address: 131.120.150.140

Destination TCP Port: 5153

Destination Data Type: Translated MEDAL Asset Messages

Status:

Figure 71. *ADS* data destination panel user interface.

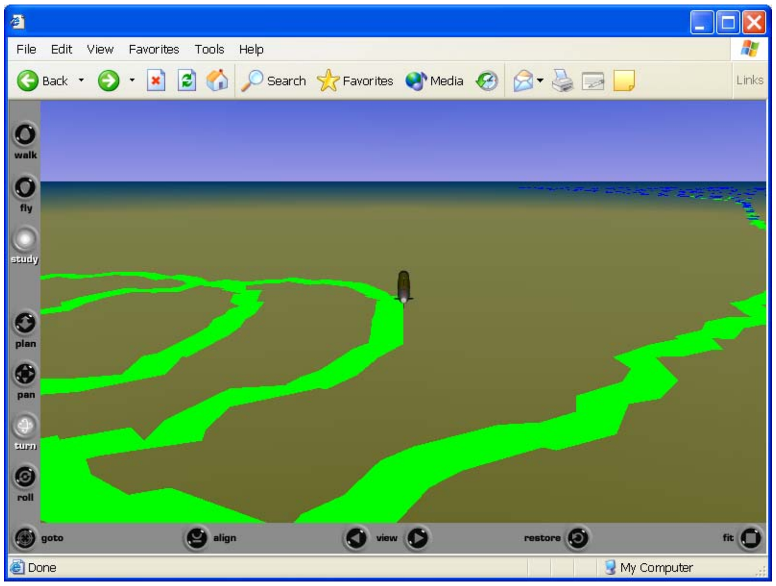


Figure 72. *ADS*-generated VRML scene from AUV data.

```
<Application name="ADS" tooltip="AUV Data Server"
             image="image/3cubes.gif" show="true">
      <Command>C:/auv/ADS/AuvDataServer.bat</Command>
      <Command>D:/auv/ADS/AuvDataServer.bat</Command>
</Application>
```

Figure 73. *ADS* entry in the AUV Workbench configuration file.

L. SUMMARY

The NPS AUV Workbench has integrated years of work by students and faculty to form a stable code base whereby continued research and development can be supported. The flexibility of the Workbench has made it simple enough for day-to-day users to get started, and at the same time allowed developers to add new tools and modules with ease. During the course of this thesis, two new modules were added. One supports Recursive Ray Acoustics (RRA) visualization and was by LT Scott Rosetti, USN The other module supports the compression of mission data using XML Schema-based Binary Compression (XSBC) [Serin 2003] created by LCDR Duane Davis, USN, on.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MESSAGE EXCHANGE TECHNIQUES AND TRANSPORT PROTOCOLS

A. INTRODUCTION

The major advantages of XML for interoperability of data are its extensibility and its ability to represent all forms of data, including graphics such as Virtual Reality Modeling Language, VRML or Extensible 3D Graphics (X3D) in text format. As systems get more complex, the need to transfer binary data as part of the XML document arises. This chapter presents possible ways to efficiently package and transport both textual and binary XML-based data via Extensible Messaging and Presence Protocol (XMPP). In addition, the design and technical implementations of possible applications using this data exchange technique are discussed.

B. COMPRESSION AND DECOMPRESSION USING JAVA.UTIL.ZIP

Compression and decompression are often applied to data to reduce network traffic during transportation and improve the performance of client/server applications. Likely candidates for applying compression and decompression are text-based files such as Scalable Vector Graphics (SVG), VRML and X3D, along with uncompressed image formats, for example 24-bit image files.

This section presents a brief introduction to data compression and decompression, and shows how to compress and decompress data (in physical files and objects) efficiently and conveniently from within Java applications using the *java.util.zip* package.

While it is possible to compress and decompress data using tools such as WinZip, gzip, and Java ARchive (JAR), these tools are used as standalone applications. It is possible to invoke these tools as separate applications from within a Java application, but this is not a portable, straightforward or efficient approach. Drawbacks of launching compression applications are especially problematic if data needs to be compressed and decompressed on the fly.

The *java.util.zip* package for zip-compatible data compression provides classes to read, create, and modify ZIP and GZIP file formats. The package also provides utility classes for computing checksums of arbitrary input streams that can be used to validate input data. This package provides one interface, fourteen classes, and two exception classes. For file manipulations, there are three main classes for the manipulation of objects and two classes for data streams.

1. Zipping Files

The *java.util.zip* package provides classes for data compression and decompression. The main classes are *ZipInputStream* for reading ZIP files and *ZipOutputStream* for writing ZIP files. The *ZipInputStream* class reads ZIP files sequentially, whereas the class *ZipFile* reads the contents of a ZIP file using a random access file internally so that the entries of the ZIP file do not have to be read sequentially.

S/N	Class	Type	Description
1.	ZipEntry	Class	Represents a ZIP file entry
2.	ZipFile	Class	Used to read entries from a ZIP file
3.	ZipInputStream	Class	An input stream filter for reading files in the ZIP file format
4.	ZipOutputStream	Class	An input stream filter for writing files to the ZIP file format

Table 21. Classes for File Compression and Decompression.

a. Compressing and Archiving Data to a ZIP File

The *ZipOutputStream* can be used to compress data to a ZIP file. The *ZipOutputStream* writes data to an output stream in a ZIP format. A sample procedure is given below.

```
import java.io.*;
import java.util.zip.*;
public class ZipStreamCS {
   static final int BUFFER = 2048;
   public static void main (String argv[]) {
      try {
         BufferedInputStream origin = null;
         FileOutputStream dest = new
                          FileOutputStream("ZipStreamCS.zip");
         CheckedOutputStream checksum = new
                             CheckedOutputStream(dest, new Adler32());
         ZipOutputStream out = new
                         ZipOutputStream(new
                             BufferedOutputStream(checksum));

         byte data[] = new byte[BUFFER];

         // get a list of files from current directory
         File f = new File(".");

         // list of files to be zipped
         String files[] = {"sample.bmp", "links.txt"};
         // or retrieve all files in current directory, f.list();

         for (int i=0; i<files.length; i++) {
            System.out.println("Adding: "+ files[i]);
            FileInputStream fi = new
                            FileInputStream(files[i]);
            origin         = new BufferedInputStream(fi, BUFFER);
            ZipEntry entry = new ZipEntry(files[i]);
            out.putNextEntry(entry);
            int count;
            while((count = origin.read(data, 0, BUFFER)) != -1) {
               out.write(data, 0, count);
            }
            origin.close();
         }
         out.close();
      } catch(Exception e) {
         e.printStackTrace();
      }
   }
} // ZipStreamCS
```

Figure 74. File compression code snippet.

b. Decompressing and Extracting Data from a ZIP File

The *java.util.zip* package provides classes for data compression and decompression. The *java.util.zip* package provides a *ZipInputStream* class for reading ZIP files. Below is the sample code to perform decompression:

```
import java.io.*;
import java.util.zip.*;

public class UnZipStreamCS {
   public static void main (String argv[]) {
      try {
         final int BUFFER = 2048;
         BufferedOutputStream dest = null;
         FileInputStream fis = new
                         FileInputStream(argv[0]);
         CheckedInputStream checksum = new
                            CheckedInputStream(fis, new Adler32());
         ZipInputStream zis = new
                        ZipInputStream(new
                                    BufferedInputStream(checksum));
         ZipEntry entry;
         while((entry = zis.getNextEntry()) != null) {
            System.out.println("Extracting: " +entry);
            int count;
            byte data[] = new byte[BUFFER];

            // write the files to the disk
            FileOutputStream fos = new
                             FileOutputStream(entry.getName());
            dest = new BufferedOutputStream(fos, BUFFER);
            while ((count = zis.read(data, 0, BUFFER)) != -1) {
               dest.write(data, 0, count);
            }
            dest.flush();
            dest.close();
         }
         zis.close();
      } catch(Exception e) {
         e.printStackTrace();
      }
   }
} // UnZipStreamCS
```

Figure 75. File Decompression code snippet.

c. *ZIP File Properties*

The *ZipEntry* class describes a compressed file stored in a ZIP file. The various methods contained in this class can be used to set and get pieces of information about the entry. These methods include retrieving information such as original size, compressed size and time of compression. This class is used by the *ZipFile* and *ZipInputStream* to read ZIP files, and the *ZipOutputStream* to write ZIP files.

2. Gzipping Objects

The ZIP format is record-based, thus suitable for file-based compression, but is not suited to manipulate objects or data streams. The GZIP is more appropriate as it operates on a single stream of data making it well suited for transferring large objects over sockets. The objects are compressed before sending across the network and decompressed upon arrival at their destination.

S/N	Class	Type	Description
1.	GZIPInputStream	Class	An input stream filter for reading compressed data in the GZIP file format
2.	GZIPOutputStream	Class	An output stream filter for writing compressed data in the GZIP file format

Table 22. Classes for Object Compression.

```
// write to GZipMission object to gzipped file

import java.io.*;
import java.util.zip.*;

public class GZipSaveMission {
   public static void main(String argv[]) throws Exception {
      // create some objects
      GZipMission auv_day1 = new GZipMission("23 Mar 2004", 1, 2);
      GZipMission auv_day2 = new GZipMission("24 Mar 2004", 29, 67);

      // serialize the objects auv_day1 and auv_day2
      FileOutputStream fos   = new FileOutputStream("gzip_db");
      GZIPOutputStream gz    = new GZIPOutputStream(fos);
      ObjectOutputStream oos = new ObjectOutputStream(gz);

      oos.writeObject(auv_day1);
      oos.writeObject(auv_day2);

      oos.flush();
      oos.close();
      fos.close();
   } // main
} // GZipSaveMission
```

Figure 76. Object Compression code snippet.

```
// read from gzipped GZipMission object

import java.io.*;
import java.util.zip.*;

public class GZipReadMission {
   public static void main(String argv[]) throws Exception{

      //deserialize objects "auv1" and "auv2"
      FileInputStream fis   = new FileInputStream("gzip_db");
      GZIPInputStream gs    = new GZIPInputStream(fis);
      ObjectInputStream ois = new ObjectInputStream(gs);

      GZipMission auv1 = (GZipMission) ois.readObject();
      GZipMission auv2 = (GZipMission) ois.readObject();

      //print the records after reconstruction of state
      auv1.print();
      System.out.println("--------------------------------");
      auv2.print();

      ois.close();
      fis.close();
   } // main
} // GzipReadMission
```

Figure 77. Object Decompression code snippet.

3. Java Archive (JAR) Format

The JAR format is based on the standard ZIP file format but adds an optional manifest file. The *java.util.jar* package provides classes for reading and writing JAR files. Using the classes provided by the *java.util.jar* package is similar to using the classes provided by the *java.util.zip* package shown earlier. A sample manifest file is given in Figure 78.

```
Manifest-Version: 1.0
Main-Class: AMVW
Class-Path: ../lib/smack.jar ../lib/smackx.jar ../lib/xercesImpl.jar
../lib/xml-apis.jar ../lib/xmlParserAPIs.jar ../lib/xalan.jar
%JAVA_HOME%/jre/lib/ext/vecmath.jar %JAVA_HOME%/jre/lib/ext/j3dcore.jar
%JAVA_HOME%/jre/lib/ext/j3dutils.jar ../lib/dis-java-vrml.jar
../lib/Jama.jar ../lib/j3d-org.jar ../lib/xj3d-all.jar ../lib/uri.jar
../lib/js.jar ../lib/vlc_uri.jar ../lib/httpclient.jar
max-heap-size: 256m
```

Figure 78. Sample *Manifest.mf* file for Java Archive.

4. Checksums

Checksums can be used to mask corrupted files or messages. Once a compressed file has been transferred to the remote machine, the checksum value can be used to detect whether the file was corrupted during the transmission.

The *Adler32* and *CRC32* classes in the *java.util.zip* package, which implement the *java.util.zip.Checksum* interface, are used to compute the checksums required for data compression. The *Adler32* algorithm is normally preferred as it faster than the *CRC32* and it is as reliable. The *getValue()* and *reset()* methods are provided to access the current checksum value or reset it to the default.

S/N	Class	Type	Description
1.	Checksum	Interface	Represents a data checksum. Implemented by the classes Adler32 and CRC32
2.	Adler32	Class	Used to compute the Adler32 checksum of a data stream
3.	CheckedInputStream	Class	An input stream that maintains the checksum of the data being read.
4.	CheckedOutputStream	Class	An output stream that maintains the checksum of the data being written
5.	CRC32	Class	Used to compute the CRC32 checksum of a data stream

Table 23. Classes for Checksum.

C. BINARY TO TEXT ENCODING AND DECODING

Embedding the byte values from the binary data file into an XML document will not work due to the XML specification's valid-character restriction and character encoding and decoding as the document travels from its source to its parsing destination.

According to the XML 1.0 specification, valid character values include the following ranges of hexadecimal values: 0x9, 0xA, 0xD, 0x20-0xd7ff, 0xe000-0xfffd, and 0x10000-0x10ffff. The specification also states that all processors are required to automatically support (and detect) the UTF-8 and UTF-16 encodings. Therefore if one of these two encodings is used when serializing XML documents, there is no need for an XML declaration (unless you need to specify version or standalone information):

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- the text ‘encoding="UTF-8"’ is optional -->
```

The ISO/IEC 10646 standard published in 1993 by the International Standards Organization (ISO) specifies the encoding of characters used to convert every written language into binary form to provide compatibility between multilingual encodings and most existing software applications that use the ASCII standard. The ISO has defined many transformations including the UTF-8 and UTF-16 encodings.

1. Brute-Force Approach

The direct approach to solving this encoding problem converts each binary data byte into its two-character hexadecimal representation. In doing so, the 256 possible byte values are encoded using two characters from the hexadecimal character set “0-9”, “A-F”:

```
byte[] buffer = readFile(filename);
int readBytes = buffer.length;
StringBuffer hexData = new StringBuffer();
for (int i=0; i < readBytes; i++) {
   hexData.append(padHexString (Integer.toHexString(0xff & buffer[i])));
}
```

A *StringBuffer* rather than plain *String* concatenation is used to build the binary buffer's resulting character representation in order to avoid the unnecessary cost of repeatedly creating and then releasing *String* class instances. A possible way to accelerate the conversion is to use a hexadecimal number lookup table as shown below:

```
public final static String[] _hexLookupTable = { "00", "01", .. ,"fe",
"ff" };

for (int i=0; i < readBytes; i++) {
   hexData.append(_hexLookupTable[0xff & buffer[i]]);
}
```

With this approach, for each byte in the original binary file, two characters are generated in the resulting XML document. Algorithmatically, this approach is reasonably efficient. One of the disadvantages of this approach is that it wastes network bandwidth, which is an important consideration when transferring large binary data files.

2. Base-64 Encoding Approach

The next approach is the Base-64 encoding conversion. Developers have historically used this approach to encode binary data within mail messages before transporting them through mail servers that allow relatively short lines of 7-bit data units.

The Base-64 encoding algorithms is described in Request for Comments (RFC) 2045 - Multipurpose Internet Mail Extensions (MIME). To encode the data, each 3-byte sequence parcels into four 6-bit numbers. Each 6-bit number is then replaced by the corresponding US-ASCII character in the Base-64 alphabet to represent binary data and character '=' for padding (i.e., byte stream's last one or two byte portions). The character set is "A-Z, a-z, 0-9, +, and /". Carriage Return Line Feed (CRLF) characters are inserted into the output stream to keep the line lengths less than 76 characters, this line length restriction does not apply when transmitting binary data as part of an XML document.

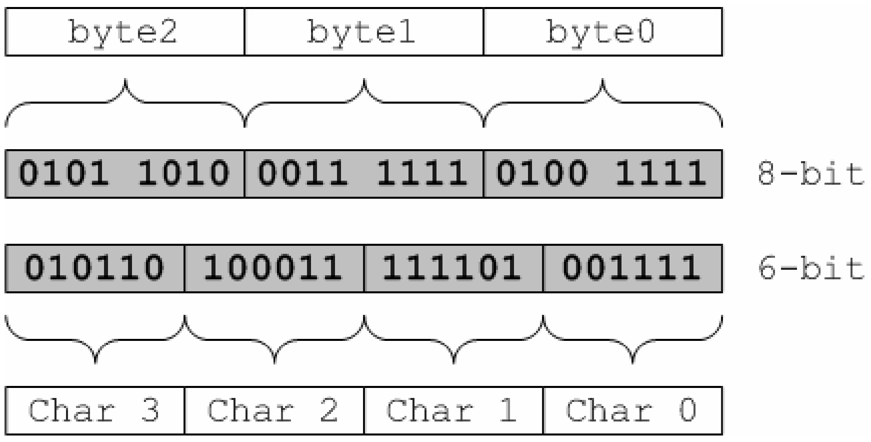


Figure 79. Base-64 encoding illustrated 3-byte stream converted to four 6-bit data units.

The advantage of this approach over the brute force method is that it encodes three data bytes using four characters resulting in an encoded document that is only 33 percent larger than the original binary document rather than 100 percent longer using the previous method. Compared to the previous approach, 1.33 characters per byte are generated instead of two characters per byte.

Another advantage of Base-64 is that it has been widely used and many implementations are available freely. As an example, this thesis uses the already available class *org.apache.soap.encoding.soapenc.Base64* in the Apache SOAP 2.3.1 implementation. In terms of conversion performance, the approach is fast since it consists of binary shift and table lookup operations. A sample base-64 encoded document is given below. The base-64 encoded content is stored in the *CDATA* section of the XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<AgentJabber>
    <AgentPayload>
<![CDATA[H4sIAAAAAAAAAO3QsU0DQRhE4Tsw4FKogBxyenIRboaABmjquDNISKsXTPZWaOb
TybDR0//2/vW53PZyWpbn/fe6f6/rsqzL+fZ+/Vj+tv5++y77tw1/XC4//2zbtnRd9z+3nee
jBwA9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0AOAHgD0AKAHgPHhcYKNjXbPsd4pW++
UrXfK1jtl652y9U7ZeqdsvVO23ilb75Std8rWO2XrnbL1TtnGO81ADwB6ANADgB4A9ACgBwA
9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0AOAHgD0AKAHAD0A6AFADwB6ALjb5qMHAD0
A6AFADwB6ANADgB4A9ACgBwA9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0AOAHgD0AKA
HAD0A6AFADwDjwzrBxka751jvlK13ytY7ZeudsvVO2XqnbL1Ttt4pW++UrXfK1jtl652y9U7
Zeqds451moAcAPQDoAUAPAHoA0AOAHgD0AKAHAD0A6AFADwB6ANADgB4A9ACgBwA9AOgBQA8
AegDQA4AeAPQAoAcAPQCctvnoAUAPAHoA0AOAHgD0AKAHAD0A6AFADwB6ANADgB4A9ACgBwA
9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0APA+HA/wcZGu+dY75Std8rWO2XrnbL1Ttl
6p2y9U7beKVvvlK13ytY7ZeudsvVO2XqnbOOdZqAHAD0A6AFADwB6ANADgB4A9ACgBwA9AOg
BQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0AOAHgD0AKAHAD0APG3z0QOAHgD0AKAHAD0A6AF
ADwB6ANADgB4A9ACgBwA9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0AOAHgD0AKAHgPH
hYYKNjXbPsd4pW++UrXfK1jtl652y9U7ZeqdsvVO23ilb75Std8rWO2XrnbL1TtnGO81ADwB
6ANADgB4A9ACgBwA9AOgBQA8AegDQA4AeAPQAoAeAb02mddvKvQAA]]>
    </AgentPayload>
</AgentJabber>
```

Figure 80. Sample XML document with base-64 encoded data in CDATA section.

3. Complex and Proprietary Algorithms

For more efficient binary to text encoding, complex algorithms such as Huffman can be employed. At the same time, it is possible to treat binary to text encoding as a form of encryption through the use of a complex and proprietary encoding scheme. Similarly encryption algorithms can be introduced during the compression process. Thus, receiving clients will require a proper implementation of the encoding algorithm to decode the binary data from the XML messages necessitating the deployment of dedicated clients or libraries. This also brings into play the issues of software updates, correct implementation/program invocation and efficient deployment of new versions. This is opposed to Base-64 encoding, which is well-known and easily implemented.

In the article “Transfer binary data in an XML document” [Pentakalos 2001], the author implemented a simple Huffman encoding algorithm that uses the binary data set statistical properties to compress the encoded character stream. For many data sets, if a histogram is constructed for each byte value's occurrence frequency within the data set, an uneven distribution can be observed, where some bytes are used frequently while others rarely or not at all.

A properly customized Huffman coding can take advantage of this statistical property to reduce the average code length. Most frequently used bytes are represented in single characters or short character sequences, and the least frequently used with longer character sequences. For cases where the distribution is highly skewed towards a byte value subset, this encoding approach is effective, but it is not as effective for cases where the distribution is fairly uniform.

D. MESSAGING PROTOCOLS

1. Simple Mail Transfer Protocol (SMTP)

The objective of Simple Mail Transfer Protocol (SMTP) is to transfer mail reliably and efficiently. SMTP is independent of the particular transmission subsystem and requires only a reliably ordered data-stream channel. Mail via SMTP is a widely supported capability that can be used for messaging across firewalls.

An important feature of SMTP is its capability to relay mail across transport service environments. A transport service provides an inter-process communication environment (IPCE) that may cover one network, several networks, or a subset of a network. It is important to realize that IPCEs do not have a one-to-one relationship with networks. A process can communicate directly with another process through any mutually known IPCE. Email is one example of an application relying on IPCEs. Mail can be communicated between processes in different IPCEs by relaying through a process connected to two (or more) IPCEs. More specifically, mail can be relayed between hosts on different transport systems by a host on both transport systems.

2. File Transfer Protocol (FTP) and Secure FTP (SFTP)

The objectives of FTP are 1) to promote sharing of files, 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs. The FTP specification attempts to satisfy the diverse needs of users of maxi-hosts, mini-hosts and personal workstations, with a simple and easily implemented protocol design.

Files being transferred by FTP are vulnerable to man-in-the-middle attacks where data is intercepted and altered before sending it on its way. Various products have been developed to resolve the security problems with FTP. Some SFTP products use Secure Socket Layer (SSL) algorithm to perform the encryption, however, worth noting is that this approach should not be confused with the common use of SSL for browser-based file transfer encryptions. SSL by itself is limited in its capabilities. FTP and SFTP allow users to change directories, list directories, and grab entire batches and directories of files in one fell swoop. SSL is generally used for getting files, and is rather limited when used for putting batches of raw files in remote locations. While SSL is well-suited for short online web-based financial transactions, since it requires no special client-side software except a browser, it is not appropriate for large-scale batch file transfers due to the high computation costs (and correspondingly long delay times) of encryption/decryption.

3. HyperText Transport Protocol (HTTP) Get/Post and Secure Hypertext Transfer Protocol (HTTPS)

Currently, HTML forms allow the producer of the form to obtain information from users. These forms have proven useful in a wide variety of applications in which user input is necessary, however, this capability is limited because HTML forms do not provide a way to ask the user to submit files of data. Service providers requiring files from the user have had to implement custom user applications. Since file upload is a feature that will benefit many applications, this thesis proposes an extension to HTML to allow information providers to express file upload requests uniformly, and a MIME compatible representation for file upload responses. Also included is a description of a backward compatibility strategy that allows new servers to interact with the current HTML user agents.

HTTPS is a secure message-oriented communications protocol designed for use in conjunction with HTTP. HTTPS is designed to coexist with HTTP's messaging model and to be easily integrated with HTTP applications. Syntactically, Secure HTTP (HTTPS or S-HTTP) messages are the same as HTTP, consisting of a request or status line followed by a header and body. However, the range of headers is different and the bodies are typically cryptographically enhanced. HTTPS messages, just as the HTTP messages, consist of requests from client to server and responses from server to client.

HTTPS does not require client-side public key certificates (or public keys), as it supports symmetric key-only operation modes. This is significant because it means that spontaneous private transactions can occur without requiring individual users to have an established public key. URLs that begin with 'https' are handled using the SSL algorithm (now commonly termed as Transport Level Security - TLS) that sets up a secure, encrypted link between a Web browser and a Web server. SSL is the industry standard protocol for secure, web-based communications and transactions and is implemented as an optional protocol layer that fits between the TCP and HTTP protocol layers.

4. Messaging Queue System (e.g., Java Messaging Service)

The Java Message Service (JMS) API is an API for accessing enterprise messaging systems. It is part of the Java 2 Platform, Enterprise Edition (J2EE).

JMS is designed to make it easy to write business applications that asynchronously send and receive critical business data and events. It defines a common enterprise messaging API that can be easily and efficiently supported by a wide range of enterprise messaging products. JMS supports both message-queuing and publish-subscribe styles of messaging.

JMS messages are asynchronous requests, reports, or events that are consumed by enterprise applications, not humans. They contain vital information needed to coordinate these systems and contain precisely formatted data describing specific business actions. Through the exchange of these messages, each application tracks the progress of the enterprise.

5. Jabber/Chat Using Extensible Messaging and Presence Protocol (XMPP)

Jabber is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and other structured information in near-real time. Jabber is an open, platform independent messaging framework based on XML for real-time extensible messaging and user presence. The basic Jabber application is an instant messaging (IM) network that offers functionality similar to legacy IM services such as AIM, ICQ, MSN, and Yahoo. However, Jabber is more than just IM, and Jabber technologies offer several key advantages. The Internet Engineering Task Force (IETF) is currently formalizing the core XML streaming protocols as an approved instant messaging and presence technology under the name of XMPP. The following sections explore the use of Jabber protocol for both human and machine message passing such as agent-to-agent communications.

S/N	Protocol	Pros	Cons
1.	Email (SMTP, Microsoft mail or similar)	Open or proprietary standards. Queuing is built-in. Good for human-to-human communications.	Does not guarantee timely delivery Bad for machine-to-machine, or machine-to-human communications. Data is either stored in the message body or as an attachment. Additional processing required to extract the data from attachments. Text-only data types. Depending on applications, rich-text and HTML formats may be supported. By default, message expiry feature is not available.
2.	File Transfer Protocol (FTP) and Secure FTP	Open standards. RFC 959. Queuing is not available; i.e., no built-in resend mechanism	To prevent hackers from exploiting anonymous/guest users, anonymous/guest user IDs are turned off. User

S/N	Protocol	Pros	Cons
		to handle cases when the receiving party is not available. Near-real time performance using TCP protocol.	ID/password is either hard-coded or via operating systems' specific means of authentication. The latter is good provided the user environment is homogeneous. Message expiry not available.
3.	HyperText Transport Protocol (HTTP) Get/Post and Secure HTTP (HTTPS)	Open standards. RFC 1867. Built-in queuing mechanism. Near-real time performance using TCP protocol.	Session-less. Message expiry not available.
4.	Messaging Queue System (e.g., Java Messaging Service)	Sun provides a reference implementation for its Java Messaging Service (JMS) specifications. Built-in queuing mechanism. Support interchange of Java objects (JMS only). "Publish-subscribe" mechanism. Message expiry built-in. Supports data types. Provides methods and expectations for Quality of Service (QoS).	Vendor-specific (IBM Message Queue or Microsoft Message Queuing, MSMQ), Proprietary implementations; e.g., SonicMQ. Requires in-depth knowledge. Open-source implementation available (JBoss), but pay for support.
5.	Jabber/Chat using Extensible Messaging and Presence Protocol (XMPP)	Open standards. RFC 2779. Built-in queuing mechanism. Near-real time performance using TCP protocol. Simple "publish-subscribe" mechanism can be achieved through chat-rooms. Ease of implementation. Messages are XML-based.	Presence needs to be established at start time. Text-only data types. Workaround by embedding the data type information within the XHTML stanza. Message expiry not available.

Table 24. Comparison of messaging systems and their protocols.

E. MESSAGE REPRESENTATION

Traditional messaging systems (e.g., email) store binary data as an attachment to the message subject/body. With XMPP, all messages are XML-based making it necessary to find ways to send binary data via this protocol. The XMPP protocol includes a base protocol and many optional extensions typically documented as Jabber Enhancement Proposals (JEPs).

1. Jabber Enhancement Proposals

a. Private Data (JEP-49)

JEP-49 is a mechanism to allow users to store arbitrary XML data on an XMPP server. Each private data chunk is defined by an element name and XML namespace. Example private data:

```
<color xmlns="http://example.com/xmpp/color">
    <favorite>blue</blue>
    <leastFavorite>puce</leastFavorite>
</color>
```

A Jabber client can store any arbitrary XML on the server side by sending an *iq* chunk of type "set" to the server with a *query* child scoped by the *jabber:iq:private* namespace. The *query* element may contain any arbitrary XML fragment as long as the root element of that fragment is scoped by its own namespace. The data can then be retrieved by sending an *iq* of type "get" with a *query* child scoped by the *jabber:iq:private* namespace, which in turn contains a child element scoped by the namespace used for storage of that fragment. Using this method, Jabber entities can store private data on the server and retrieve it whenever necessary. The data stored might be anything, as long as it is valid XML. One typical usage for this namespace is the server-side storage of client preferences.

b. Extensible HyperText Markup Language (XHTML) (JEP-71)

The JEP-71 proposal defines an adaptation of XHTML 1.0 to provide alternative formatting for a text message. It provides the ability to send and receive formatted messages using XHTML. This pattern is familiar from email, wherein the HTML-formatted version of the message supplements, but does not supersede the text-only version of the message. In Jabber communications, the meaning (as opposed to formatting) of the message must always be represented as best as possible in the normal *body* child of the *message* element. Formatting is then provided by the XHTML representation of the message content within a *html* wrapper element.

```
<message>
  <body>hi</body>
  <html xmlns='http://jabber.org/protocol/xhtml-im'>
    <body xmlns='http://www.w3.org/1999/xhtml'>
      <h1>hello</h1>
    </body>
  </html>
</message>
```

These two JEPs provide two possible ways to package binary data:

- Embed a hyperlink to binary data via out-of-band (*oob*) messages.
- Embed the binary data in a *CDATA* section.

Even though these two methods could be used to represent binary data, one is not necessarily a good substitute for another. Alternatively, these methods can be applied to complement each other.

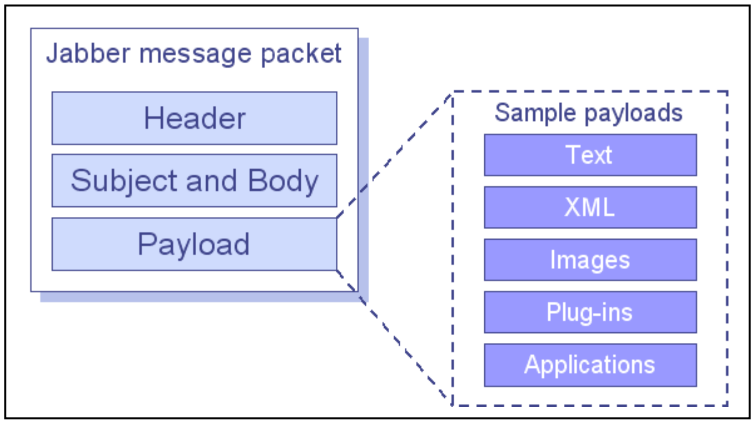
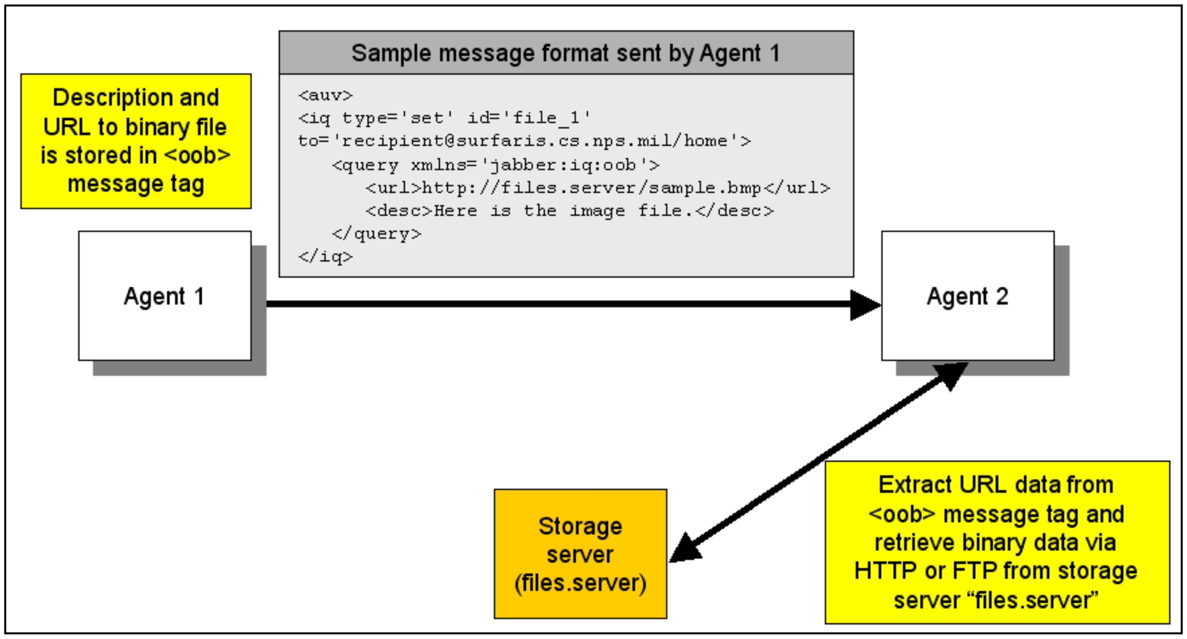


Figure 81. Packaging binary data in a Jabber message.

2. Embed Hyperlink to Binary Data via Out-of-band (oob) Messages

An out-of-band message is a message *x* extension that is embedded in a standard Jabber *message* packet (usually a message of type normal). An *oob* message contains information, typically a *url* link that clients can use to conduct direct application-to-application data transfer that bypasses the normal Jabber message routing via the Jabber server. The link typically points to a web or FTP server.


Description and URL to binary file is stored in <oob> message tag
Sample message format sent by Agent 1
Agent 1
Agent 2
Storage server (files.server)
Extract URL data from <oob> message tag and retrieve binary data via HTTP or FTP from storage server "files.server"

```
<auv>
<iq type='set' id='file_1'
to='recipient@surfaris.cs.nps.mil/home'>
   <query xmlns='jabber:iq:oob'>
      <url>http://files.server/sample.bmp</url>
      <desc>Here is the image file.</desc>
   </query>
</iq>
```

Figure 82. Overview on file transfer using *out-of-band* (*oob*) message.

Out-of-band messages are typically used to arrange transfer of large files that are impractical to route via the server. A sample *oob* message is given below:

```
<iq type='set' id='file_1' to='recipient@surfaris.cs.nps.mil/home'>
   <query xmlns='jabber:iq:oob'>
      <desc>Here is the image file u requested.</desc>
      <url>http://files.nps.mil/sample.bmp</url>
   </query>
</iq>
```

Although this method reduces network traffic on the Jabber server (note: actual traffic required to transport the binary data from one location to another is still the same), it introduces a point of failure to the transport mechanism; if the storage server is offline, there is no way to retrieve the data. This problem can be circumvented through the introduction of multiple hyperlinks to the same binary data; i.e., each hyperlink pointing to different storage locations of the same file.

```
<iq type='set' id='file_1' to='recipient@surfaris.cs.nps.mil/home'>
   <query xmlns='jabber:iq:oob'>
      <desc>Here is the image file u requested.</desc>
      <url>http://fileserver1.nps.mil/sample.bmp</url>
      <url>http://fileserver2.nps.mil/sample.bmp</url>
      <url>http://fileserver3.nps.mil/sample.bmp</url>
   </query>
</iq>
```

3. Embed Binary Data in *CDATA* Section

In order to reduce the time required to retrieve data from another location, binary data can be embedded within the XML document. This allows the recipient to process the data immediately upon receipt. By making use of compression and base-64 encoding techniques discussed earlier, it is possible to reduce the size and at the same time embed binary data in an XML message. To add another layer of reliability to this method, multiple hyperlinks to storage locations are added. This is similar to the technique discussed above. If the encoded/compressed data was corrupted during its delivery, the recipient will still be able to retrieve it from a storage server via FTP or HTTP. (Note: Since some firewalls block out FTP traffic, therefore it is better to include links to both FTP and HTTP servers.) At the same time, a maximum file size limit can be introduced to prevent overloading the Jabber server. Thus for a large file (e.g., larger than 1MB), the *CDATA* section will not be populated with the encoded binary data, and hyperlinks will be used to retrieve the file. Upon receipt, the client spawns an HTTP or FTP process to retrieve the file from the storage location. A sample XML message (without the Jabber message wrapper) and the function to perform base-64 encoding and compression are given Appendix E.

```
<AgentJabber>
    <!--Payload 1-->
    <AgentPayload checksum="1234567"
        content-transfer-encoding="base-64"
        content-type="application/x-zip-compressed"
        desc="The filename is [GAMMA.bmp]"
        filename="GAMMA_apple.bmp"
        filesize="48586"
        timestamp="20040026114827">
<![CDATA[H4sIAAAAAAAAAO3QsU0DQRhE4Tsw4FKogBxyenIRboaABmjquDNISKsXTPZWaOb
TybDR0//2/vW53PZyWpbn/fe6f6/rsqzL+fZ+/Vj+tv5++y77tw1/XC4//2zbtnRd9z+3nee
jBwA9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0AOAHgD0AKAHgPHhcYKNjXbPsd4pW++
UrXfK1jtl652y9U7ZeqdsvVO23ilb75Std8rWO2XrnbL1TtnGO81ADwB6ANADgB4A9ACgBwA
oA0APA+HA/wcZGu+dY75Std8rWO2XrnbL1Ttl6p2y9U7beKVvvlK13ytY7ZeudsvVO2XqnbO
OdZqAHAD0A6AFADwB6ANADgB4A9ACgBwA9AOgBQA8AegDQA4AeAPQAoAcAPQDoAUAPAHoA0A
OAHgD0AKAHAD0APG3z0QOAHgD0AKAHAD0A6AFADwB6ANADgB4A9ACgBwA9AOgBQA8AegDQA4
AeAPQAoAcAPQDoAUAPAHoA0AOAHgD0AKAHgPHhYYKNjXbPsd4pW++UrXfK1jtl652y9U7Zeq
dsvVO23ilb75Std8rWO2XrnbL1TtnGO81ADwB6ANADgB4A9ACgBwA9AOgBQA8AegDQA4AeAP
QAoAeAb02mddvKvQAA]]>
    <url>http://www.google.com/images/logo.gif</url>
    </AgentPayload>
</AgentJabber>
```

Figure 83. Sample XML message with encoded binary data.

S/N	Name	Type	Description
1.	AgentJabber	Element	Root.
2.	AgentPayload	Element	One or many <AgentPayload> elements under <AgentJabber>
3.	checksum	Attribute	Numeric checksum value for the byte data stored within CDATA section.
4.	content-transfer-encoding	Attribute	Technique used to encode binary data in CDATA section. Valid values are "base64" and "Huffman".
5.	content-type	Attribute	Details of binary data that allows the correct application to process it.
6.	desc	Attribute	File description.
7.	filename	Attribute	Original file name.
8.	filesize	Attribute	Original file size in bytes.
9.	timestamp	Attribute	Date/time value of file creation in "yyyymmddhhmmss" format; e.g., 20040115235959 is 15 Jan 2004 at time 23:59:59.
10.	CDATA	Element	Base64 encoded binary data.
11.	url	Element	Defines the location where the binary file/data can be found

Table 25. XML tagset to define the XHTML payload in the Jabber message.

F. DESIGN AND IMPLEMENTATION

1. Overview

This section explores possible ways to implement chat clients that enable the data exchange of both messages and binary files using Jabber protocol. The three approaches covered in this section are: web-based, standard client and customized Jabber client. This section discusses how an application that makes use of Jabber protocol for communications can be implemented. Technical details on the use of compression and base-64 encoding to facilitate message exchange of both textual (e.g., mission scripts) and non-textual (e.g., images) data are also covered.

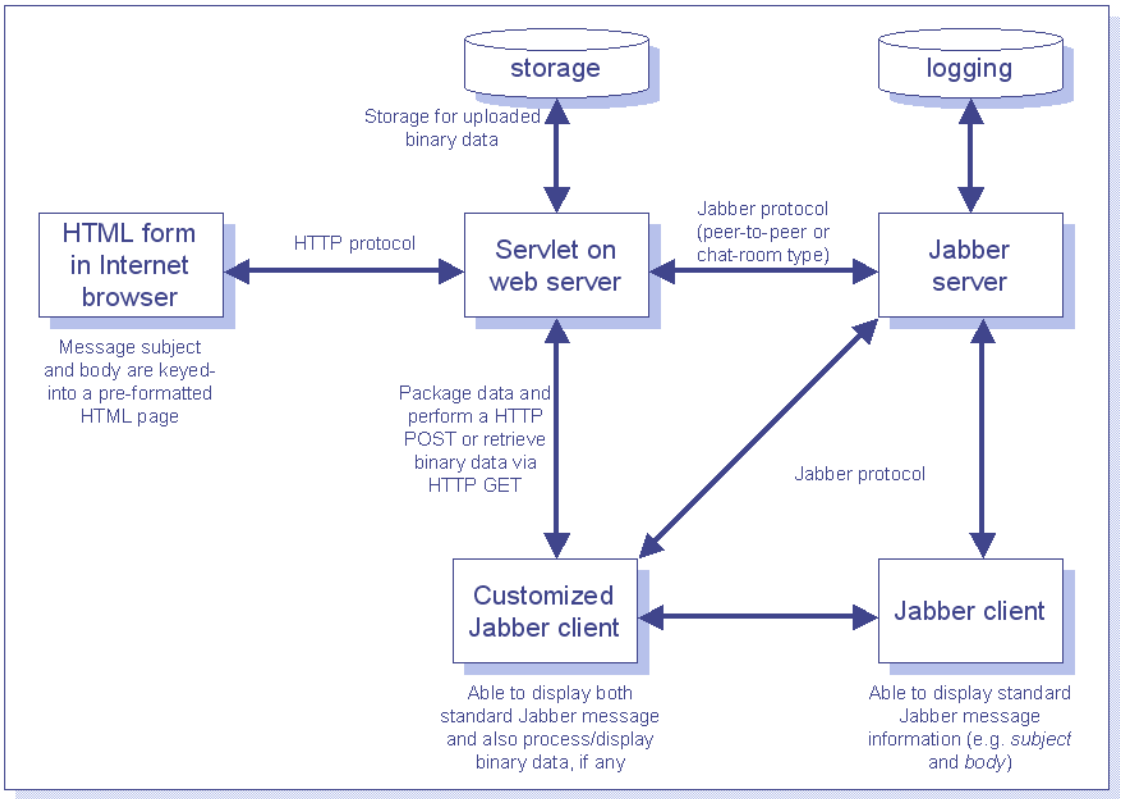

storage
logging
Storage for uploaded binary data
HTML form in Internet browser
HTTP protocol
Servlet on web server
Jabber protocol (peer-to-peer or chat-room type)
Jabber server
Message subject and body are keyed-into a pre-formatted HTML page
Package data and perform a HTTP POST or retrieve binary data via HTTP GET
Jabber protocol
Customized Jabber client
Jabber client
Able to display both standard Jabber message and also process/display binary data, if any
Able to display standard Jabber message information (e.g. subject and body)

Figure 84. Overview of the three approaches to Jabber instant messaging.

2. Introduction to Jabber Protocol

A full Jabber ID takes the form *[user name]@[Jabber server]/[resource]*, similar to an email address. A groupchat/chat room takes the form: *[chatroom/group name]@[Jabber groupchat server]/[nickname]*. There are three core Jabber protocols, namely:

- Message. This is responsible for the delivering of data. Most of the time, it accounts for the bulk of the packet traffic on the Jabber network. These messages can resemble full-scale email messages or form line-by-line messages in chat sessions. This protocol uses the *message* packet. A sample *message* packet is given below:

```
<message xml:lang="en-us"
       to="savage@conference.xchat.movesinstitute.org"
       type="groupchat">
<body>This is a test message</body>
</message>
```

Figure 85. A sample "groupchat" message to "savage" chatroom.

```
<message xml:lang="en-us"
       to="auvrobot@surfaris.cs.nps.navy.mil"
       type="chat">
       <subject>This is the subject</subject>
       <body>This is a test message</body>
</message>
```

Figure 86. A sample chat message to "auvrobot" Jabber user.

S/N	Name	Type	Description
1.	message	Element	Application to be invoked from toolbar.
2.	xml:lang	Attribute	Language used.
3.	to	Attribute	Receiver of message packet, i.e., another Jabber user or Jabber server.
4.	type	Attribute	An enumeration to indicate message type. Possible values are `groupchat` and `chat`.
5.	subject	Element	Message subject. This is available for `chat` messages only.
6.	body	Element	Message body.

Table 26. Message packet types and protocol.

- Presence. The basic presence protocol is used in presence update and presence subscription management. Presence update is to inform people of the user's current presence state. Presences subscription management allows people to subscribe to another user's presence update packet and control who has access to their own presence. This protocol uses the *presence* packet. A sample *presence* packet is given below:

```
<presence from="auvrobot@surfaris.cs.nps.navy.mil"
          to="savage@conference.xchat.movesinstitute.org"
          type= "available">
      <status>I am now logged on to chatroom.</status>
      <priority>10</priority>
      <show>chat<show>
</presence>
```

Figure 87. A sample "presence" packet from "auvrobot" to "savage groupchat" server.

S/N	Name	Type	Description
1.	Presence	Element	Application to be invoked from toolbar.
2.	from	Attribute	Sender of presence packet.
3.	to	Attribute	Receiver of presence packet, normally a Jabber server.
4.	type	Attribute	An enumeration to indicate presence status. Possible values are "available", "unavailable", "subscribe", "un subscribe", "subscribed", "un subscribed" and "error".
5.	status	Element	User-definable free-form text description to be displayed.
6.	priority	Element	Non-negative integer value to delivery priority for this current resource. Higher numbers have higher priority.
7.	show	Element	Jabber clients typically use this to display presence icons, sound or alerts. If no `show` state is indicated, the user is in `normal` or `online` state. The other possible states are `chat`, `away`, `xa` (extended away) and `dnd` (do not disturb).

Table 27. "Presence" packet types and protocol.

- Info/Query (IQ). This handles everything else that does not fall under a Message or Presence packet. It serves as a catch-all protocol. If a protocol is not sending a message, or managing presence, it is an IQ protocol. IQ is a generic request-response protocol and it is designed to be easily extensible with IQ extension protocols. An IQ packet may look like the following:

```
<iq type='get' to='handlerJID'>
      <query xmlns='jabber:iq-auth'>
            <username>auvcontrol</username>
      </query>
</iq>
```

A typical Jabber session:

- Connect with a Jabber server (e.g., *xchat.moveinstitute.org*).
- Open a Jabber stream by logging in using user account "*auvcontrol*", password "*auvpwd*" on domain "*xchat.moveinstitute.org*".
- Update presence status to "available".

3. Web-based Jabber Client

The components required for this approach are a HTML form running on the client's Internet browser and a Java servlet on the web server to process, package and send out the Jabber messages. An HTML form can be easily implemented using standard HTML objects. Below are some examples of HTML objects used to generate the HTML form:

```
<!-- Creates a single-line text entry control -->
<input type="text" name="edtSubject" value="Message Subject" size="30">
```

```
<-- Creates a file upload object with a text box and Browse button. -->
<input type="file" name="edtFile">
```

For a file upload to take place:

- The *INPUT type=file* element must be enclosed within a `FORM` element.
- A value must be specified for the *NAME* attribute of the *INPUT type=file* element.
- The *METHOD* attribute of the *FORM* element must be set to post.
- The *ENCTYPE* attribute of the *FORM* element must be set to multipart/form-data.

To handle a file upload, a server-side process must be running that can handle multipart/form-data submissions. For this thesis, a Java servlet was implemented to parse and package the uploaded HTML data to be sent out as an XHTML Jabber message. The servlet uses the O'Reilly multipart file upload library (*com.oreilly.servlet*) to extract the uploaded file data.

With a HTML form, user input can be validated before it is sent to the server. If validation is performed on the client web browser, it is likely to be implemented using Netscape's JavaScript or Microsoft's VBScript. JavaScript syntax is different from VBScript. Since JavaScript and VBScript are specifically designed to work in browsers, they do not include features that are normally outside the scope of scripting, such as file access and printing. JavaScript is preferred for validation code due to compatibility and consistency issues, especially VBScript on different browsers. Specifically, while VBScript is fully supported by Microsoft Internet Explorer, the same cannot be said for other browsers.

At the time of this thesis, a "send-only" HTML form has been developed. Therefore the user will still require a Jabber client to view incoming Jabber messages. A possible way to implement a message receive module in the Internet browser is as follows.

- Modify the servlet to listen for incoming messages from one or more chat-rooms or JIDs.
- Upon receipt of Jabber messages, convert them to HTML format (include date/time for easy reference). This is to be displayed within a HTML frame on the client Internet browser.
- HTML is for display purposes, it does not have the means to actively listen for updated web pages. Thus a simple polling mechanism is required. This example causes the browser to reload the document every five seconds, but it causes unnecessary refresh on the web page.

```
<meta http-equiv="refresh" content="5; ">
```

Steps involved in sending a message via the web-based Jabber approach:

```
      Client:
•     Access to Jabber-enabled website using standard Internet browser.
•     Key-in required data in the HTML Form.

      Server:
•     Servlet receives posted HTML data and files, if any.
•     Re-package HTML data to XHTML Jabber format.
•     Get list of recipients from configuration file.
•     Log into Jabber server.
•     Send packaged data as Jabber message to recipients.
```

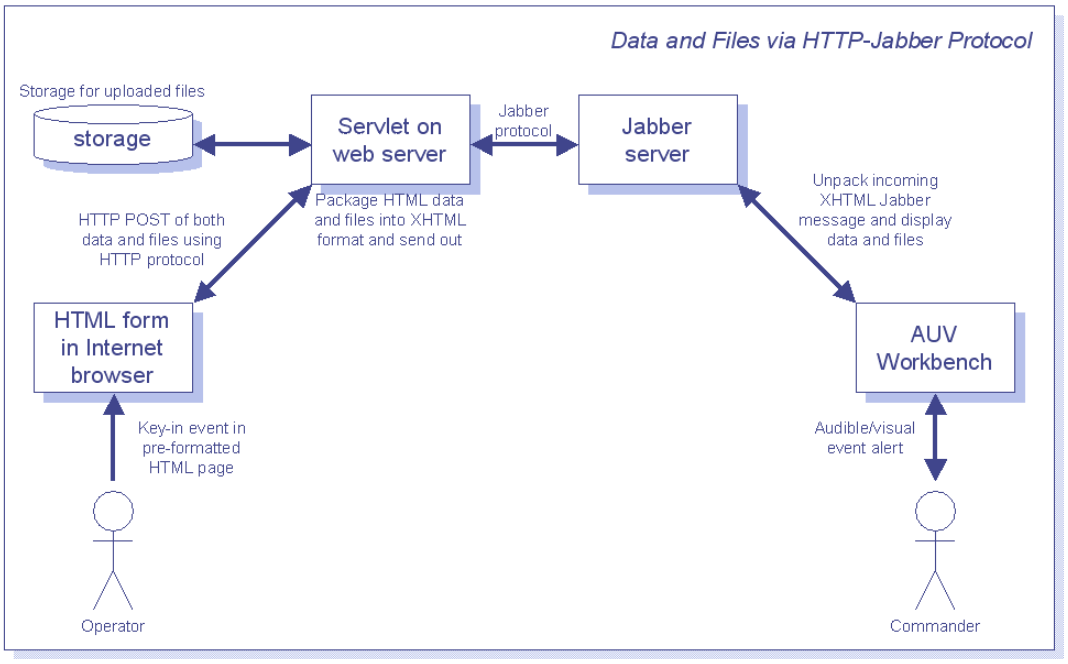
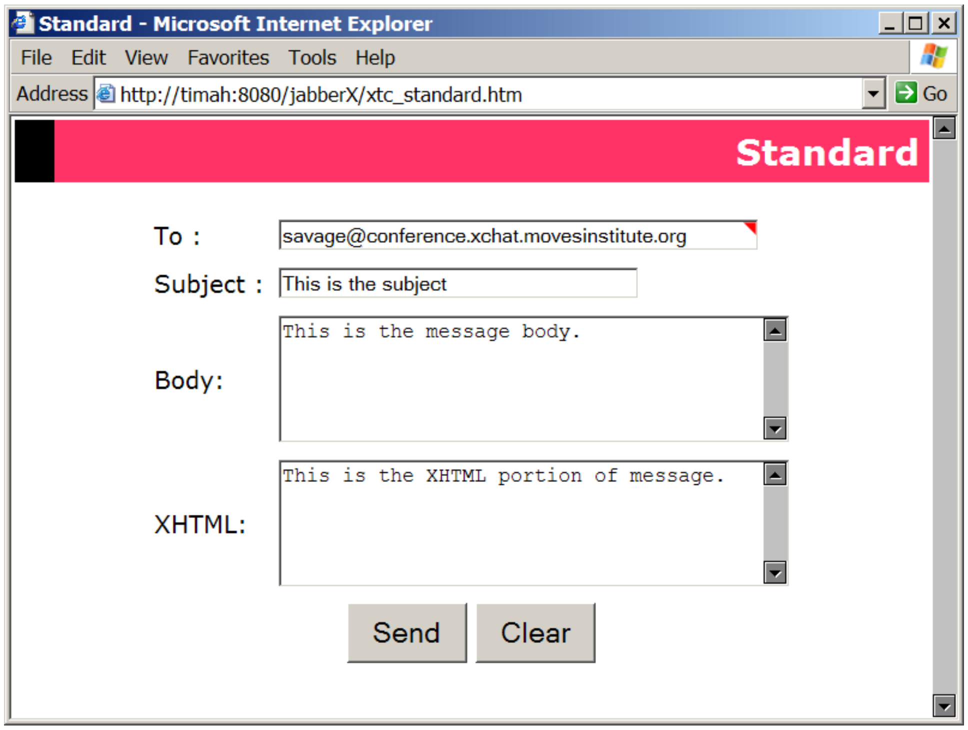

Data and Files via HTTP-Jabber Protocol
Storage for uploaded files
storage
Servlet on web server
Jabber protocol
Jabber server
Package HTML data and files into XHTML format and send out
HTTP POST of both data and files using HTTP protocol
Unpack incoming XHTML Jabber message and display data and files
HTML form in Internet browser
AUV Workbench
Key-in event in pre-formatted HTML page
Audible/visual event alert
Operator
Commander

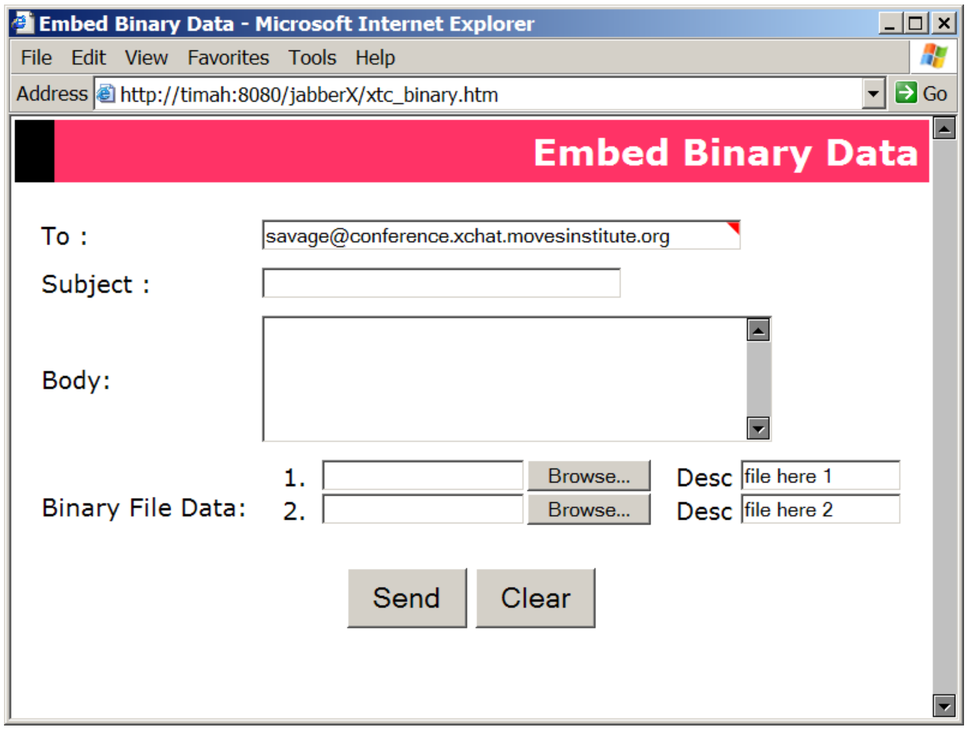
Figure 88. Data and file transfer via HTTP-Jabber protocol.


Standard - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
Standard
Send
Clear

http://timah:8080/jabberX/xtc_standard.htm

To :	savage@conference.xchat.movesinstitute.org
Subject :	This is the subject
Body:	This is the message body.
XHTML:	This is the XHTML portion of message.

Figure 89. Sample HTML form for posting of data.

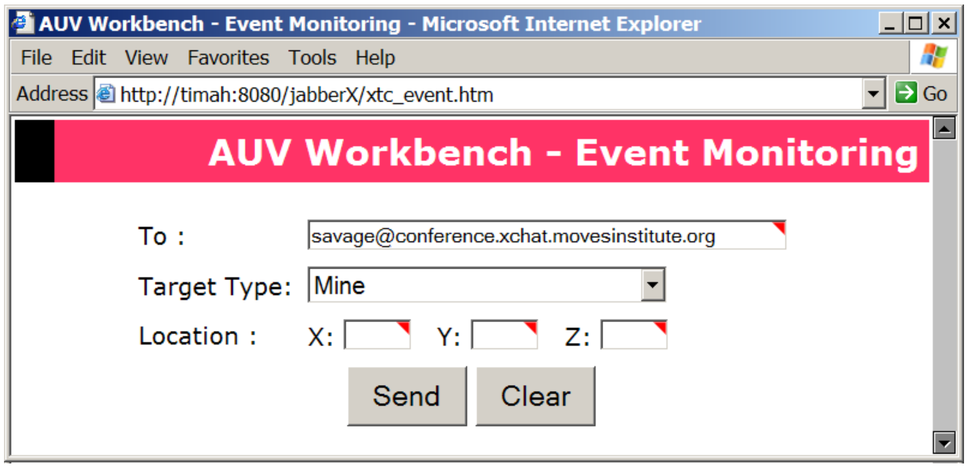

Embed Binary Data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Go
Browse...
Browse...
Send
Clear

Address http://timah:8080/jabberX/xtc_binary.htm

Embed Binary Data

To :	savage@conference.xchat.movesinstitute.org
Subject :	
Body:	
Binary File Data:	1. Desc file here 1
	2. Desc file here 2

Figure 90. Sample HTML form for posting of data and files.


AUV Workbench - Event Monitoring - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Go
Send
Clear

Address http://timah:8080/jabberX/xtc_event.htm

AUV Workbench - Event Monitoring

To :	savage@conference.xchat.movesinstitute.org
Target Type:	Mine
Location :	X: Y: Z:

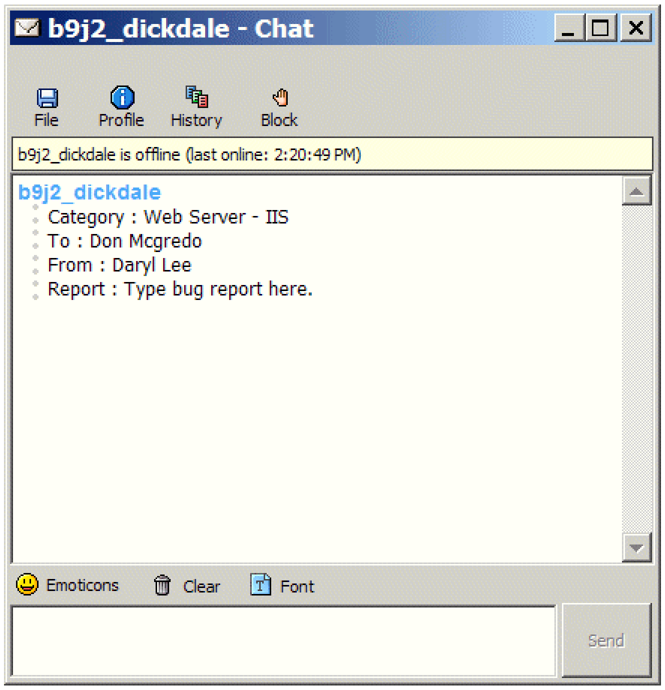
Figure 91. Sample HTML form for Target Events.

4. Standard Jabber Client

Standard Jabber clients are only able to send and receive standard Jabber messages (subject and message body). Certain clients (e.g., Rhymbox) have a “Console” interface that allows the user to key-in and send customized Jabber messages, however the files still need to be compressed, binary-to-text encoded and packaged as XHTML format before they can be sent out. The user needs to manually invoke the modules (if any) that will perform the conversion, copy-and-paste the textual results into the "Console" interface and then send it out. This process is tedious and diminishes the usability of this technique of message exchange, especially when the load increases.

Steps involved in sending a message via the standard Jabber client approach:

```
        Client:
•       Log into Jabber server.
•       Key-in message subject and body.
•       Send to specific user IDs or chat-rooms.
```



b9j2_dickdale is offline (last online: 2:20:49 PM)

b9j2_dickdale
Category : Web Server - IIS
To : Don Mcgredo
From : Daryl Lee
Report : Type bug report here.

Figure 92.   Rhymbox Jabber client.

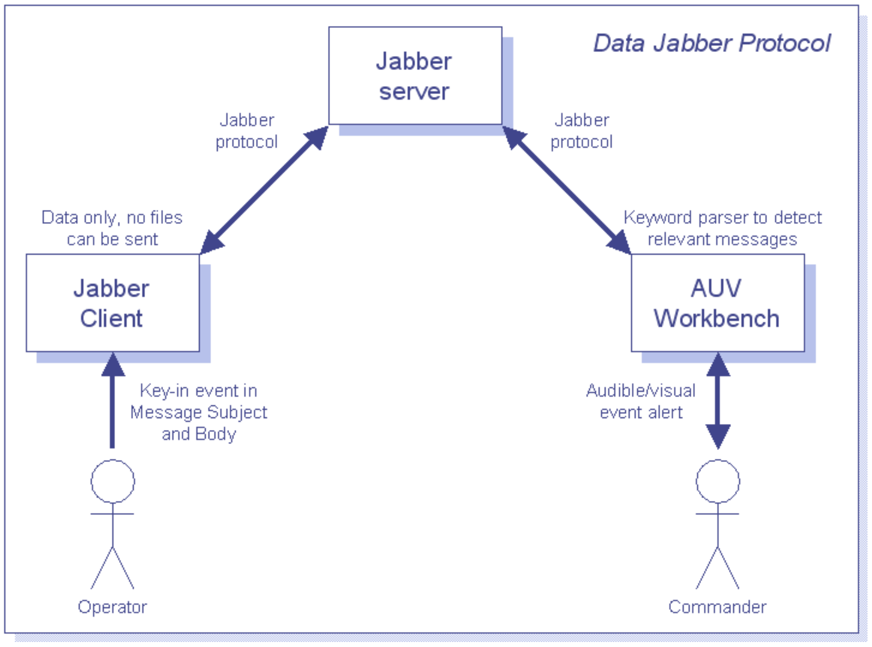

Data Jabber Protocol
Jabber server
Jabber protocol
Jabber protocol
Data only, no files can be sent
Keyword parser to detect relevant messages
Jabber Client
AUV Workbench
Key-in event in Message Subject and Body
Audible/visual event alert
Operator
Commander

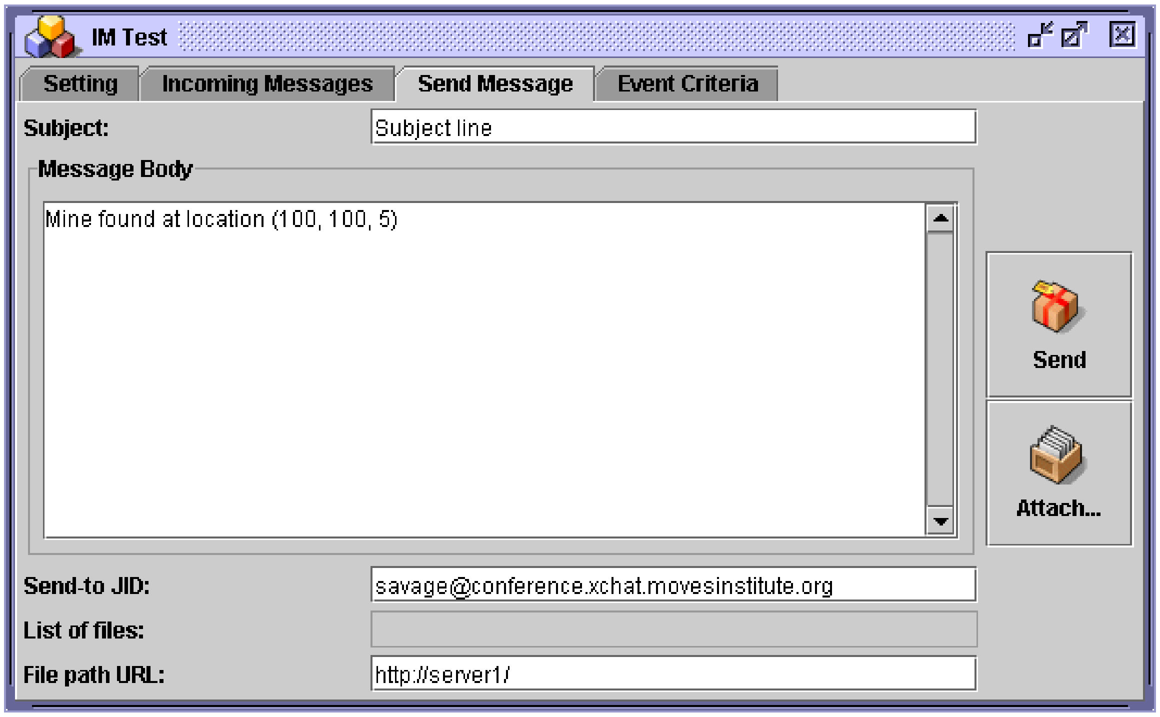
Figure 93. Data exchange using standard Jabber client.

5. Customized Jabber Client

A customized Jabber client can send and receive standard Jabber messages as well as messages with embedded binary data. The customized client automatically packages and sends out XHTML messages. It is also able to mimic an Internet browser and perform a HTTP POST of data and files to a Jabber-enabled web site (i.e., deployed with the servlet to receive posted data and files). This ability has been incorporated in the customized Jabber client for this thesis.

Of the three proposed solutions, only the customized client is able to re-generate the binary data either by decompressing and decoding the embedded data, or by retrieving the file from a hyperlink (specified within *url* tags). Once the data has been re-generated, it invokes the module or application to display the data.

The third approach functions both as a backup for Jabber communications and as a storage location for binary data that is referenced (in *url* tags) in the Jabber XHTML messages. Although the customized Jabber client is the most complex and takes the longest to develop, it is the most flexible solution. To enable reuse, the customized client that has been developed for this thesis is generic. Therefore it can be plugged into any application for that requires text and binary data message passing.



Subject: Subject line

Message Body

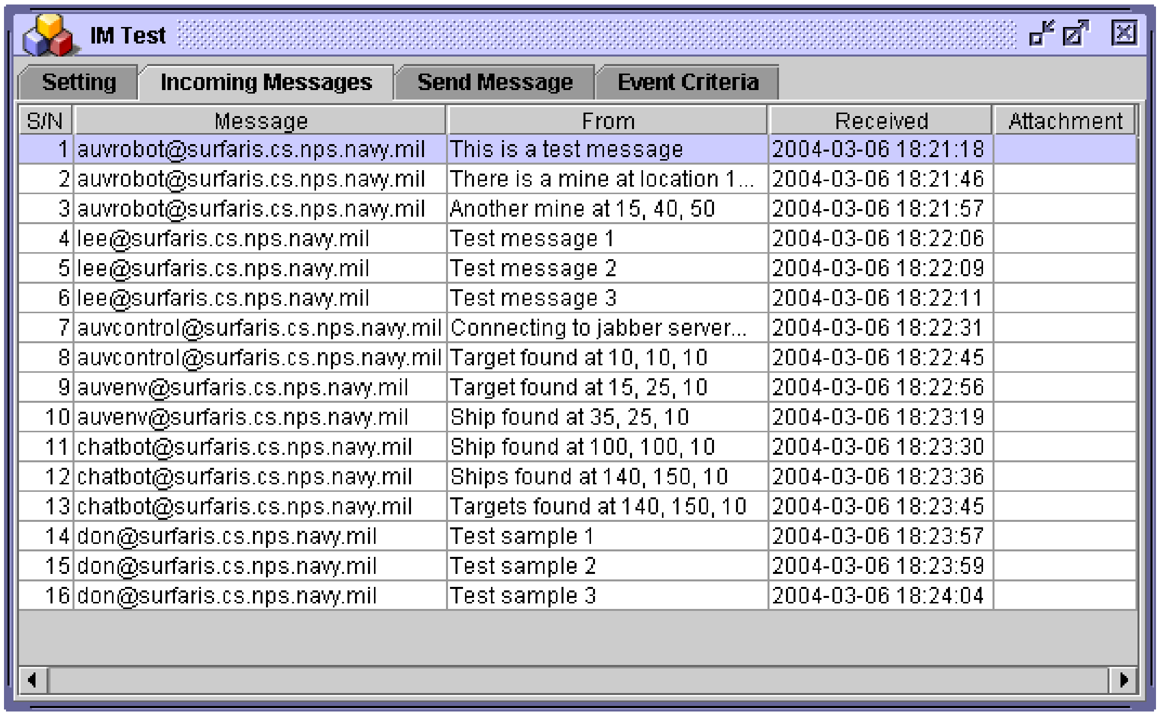
Mine found at location (100, 100, 5)

Send-to JID: savage@conference.xchat.movesinstitute.org

List of files:

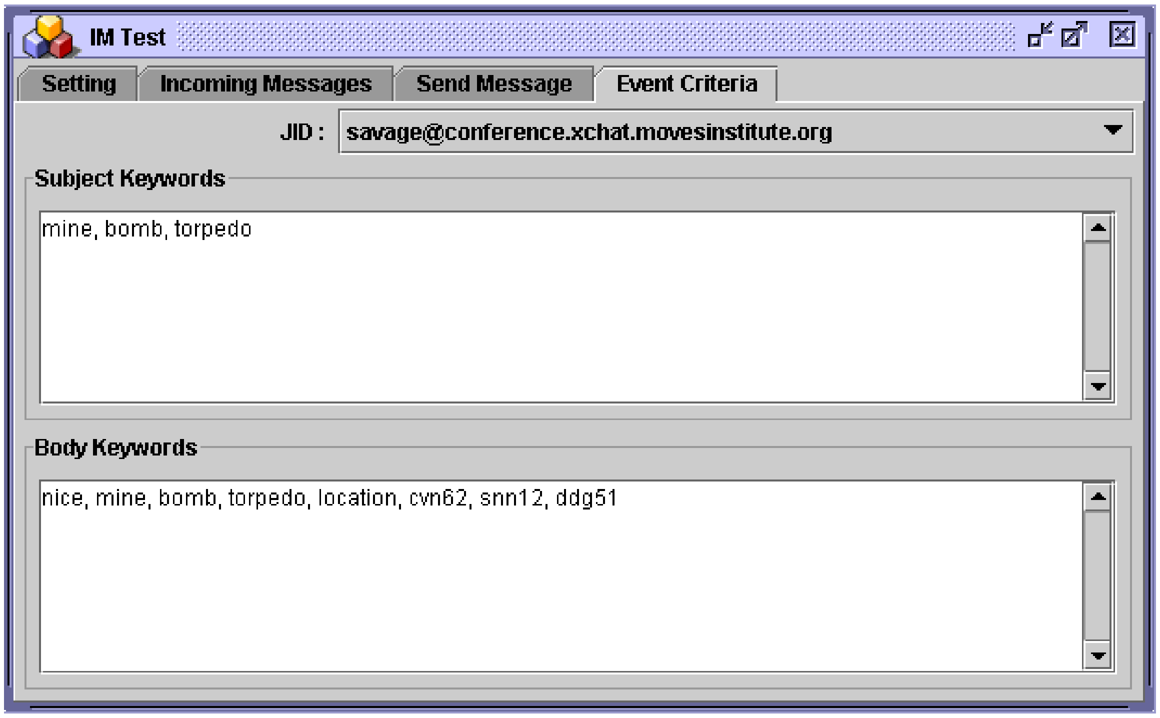
File path URL: http://server1/

Figure 94. Customized Jabber client user interface to send data and files as Jabber message.



S/N	Message	From	Received	Attachment
1	auvrobot@surfaris.cs.nps.navy.mil	This is a test message	2004-03-06 18:21:18	
2	auvrobot@surfaris.cs.nps.navy.mil	There is a mine at location 1...	2004-03-06 18:21:46	
3	auvrobot@surfaris.cs.nps.navy.mil	Another mine at 15, 40, 50	2004-03-06 18:21:57	
4	lee@surfaris.cs.nps.navy.mil	Test message 1	2004-03-06 18:22:06	
5	lee@surfaris.cs.nps.navy.mil	Test message 2	2004-03-06 18:22:09	
6	lee@surfaris.cs.nps.navy.mil	Test message 3	2004-03-06 18:22:11	
7	auvcontrol@surfaris.cs.nps.navy.mil	Connecting to jabber server...	2004-03-06 18:22:31	
8	auvcontrol@surfaris.cs.nps.navy.mil	Target found at 10, 10, 10	2004-03-06 18:22:45	
9	auvenv@surfaris.cs.nps.navy.mil	Target found at 15, 25, 10	2004-03-06 18:22:56	
10	auvenv@surfaris.cs.nps.navy.mil	Ship found at 35, 25, 10	2004-03-06 18:23:19	
11	chatbot@surfaris.cs.nps.navy.mil	Ship found at 100, 100, 10	2004-03-06 18:23:30	
12	chatbot@surfaris.cs.nps.navy.mil	Ships found at 140, 150, 10	2004-03-06 18:23:36	
13	chatbot@surfaris.cs.nps.navy.mil	Targets found at 140, 150, 10	2004-03-06 18:23:45	
14	don@surfaris.cs.nps.navy.mil	Test sample 1	2004-03-06 18:23:57	
15	don@surfaris.cs.nps.navy.mil	Test sample 2	2004-03-06 18:23:59	
16	don@surfaris.cs.nps.navy.mil	Test sample 3	2004-03-06 18:24:04	

Figure 95. Customized Jabber client user interface to display list of incoming Jabber messages.



JID : savage@conference.xchat.movesinstitute.org

Subject Keywords

mine, bomb, torpedo

Body Keywords

nice, mine, bomb, torpedo, location, cvn62, snn12, ddg51

Figure 96. Customized Jabber client user interface – Event Monitoring Criteria.

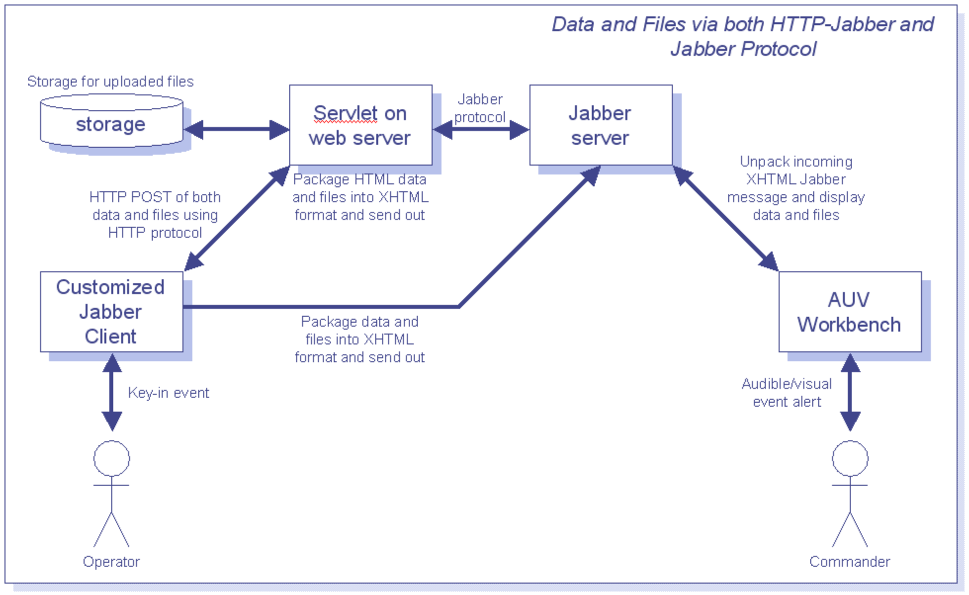

Data and Files via both HTTP-Jabber and Jabber Protocol
Storage for uploaded files
storage
Servlet on web server
Jabber protocol
Jabber server
Package HTML data and files into XHTML format and send out
HTTP POST of both data and files using HTTP protocol
Unpack incoming XHTML Jabber message and display data and files
Customized Jabber Client
Package data and files into XHTML format and send out
AUV Workbench
Key-in event
Audible/visual event alert
Operator
Commander

Figure 97. Data and file transfer via HTTP-Jabber and Jabber protocol.

The following table provides an overview of the three techniques discussed. For this thesis, emphasis was placed on the customized client due to the need to integrate into the AUV Workbench. The web-based solution using HTML form is also a viable solution provided there is a way to perform "smart" refresh of posted messages.

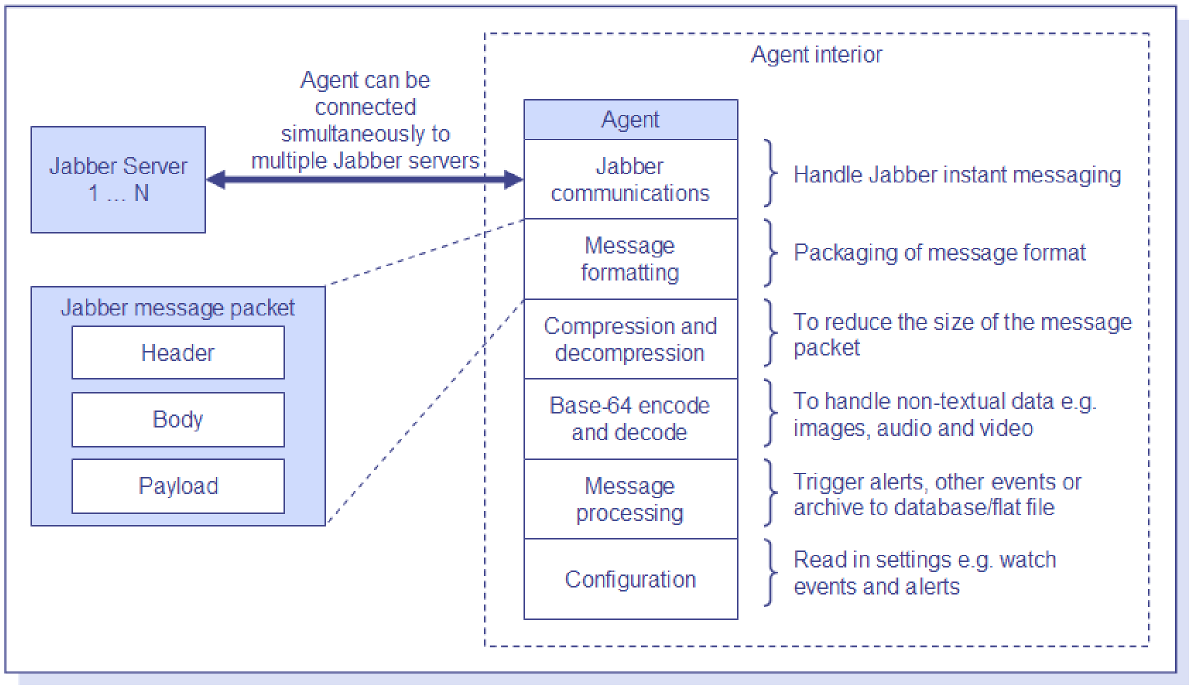
S/N	Description	Web-based	Standard Client	Customized Client
1.	Components.	HTML Form on client Internet browser for posting of HTML data. Java servlet on a web server to process uploaded data and communications with Jabber server.	Standard client. Jabber server communications.	Customized client. Message processing and Jabber communications.
2.	Message type.	Standard subject-body message and complex messages with embedded binary data (able to encode only).	Standard subject-body message.	Standard subject-body message and complex messages with embedded binary data.
3.	Message validation.	Client-side using JavaScript or at server-end.	None.	Customized user interface allows for complex data validation.
4.	Protocol.	HTTP-to-Jabber.	Jabber only.	Jabber and HTTP-to-Jabber. The web-based solution can serve as a backup.
5.	Direction of communications.	One-way – Able to send message only.	Send and receive messages.	Send and receive messages.
6.	Recipients per message	Able to specify multiple recipients.	One recipient at a time.	Able to specify multiple recipients.
7.	Performance.	Compared to the other two, this is slower since it is going through a web server.	Near-real time.	Near-real time.
8.	Deployment.	Minimal or none. As long as the client has access to the web site.	Required. Periodic software update.	Required. Periodic software update.
9.	Uses.	Report submission. Posting of happening/events.	Human-human interactions.	Human-human Machine-machine and human-machine interactions.

Table 28. Comparison of the three approaches.

6. Interior of a Jabber-enabled Agent

This section provides an overview of how the different pieces of technologies are put together, using agents as an exemplar. Each agent makes use of its own unique Jabber ID to identify itself within the Jabber network. There are two ways whereby the agents can communicate. Peer-to-peer "chat" messages are used for dedicated agent-to-agent communication; whereas “groupchat” messages are used to allow multiple agents to listen to and react to messages posted on the chatroom. The “groupchat” feature is similar to a “publish-subscribe” mechanism.

Once the binary data has been extracted, it can be worked on by other processes or saved in a database for archival purposes. Components of the agent interior are shown in Figure 98 and described below.



Agent interior
Agent can be connected simultaneously to multiple Jabber servers
Jabber Server 1 … N
Agent
Jabber communications
Handle Jabber instant messaging
Message formatting
Packaging of message format
Jabber message packet
Header
Body
Payload
Compression and decompression
To reduce the size of the message packet
Base-64 encode and decode
To handle non-textual data e.g. images, audio and video
Message processing
Trigger alerts, other events or archive to database/flat file
Configuration
Read in settings e.g. watch events and alerts

Figure 98. Interior of a Jabber-enabled agent.

a. *Jabber Communications*

This handles network communications for the Jabber protocol such as login, joining chatrooms, initiating chat sessions and listening for messages.

b. *Message Formatting*

Packaging of the message header and its payload is done in this module. The message payload includes message subject, body and binary data, if any. This module is responsible for packaging one or multiple files within the same Jabber message. An example with two files is show in Figure 99.

```
<?xml version="1.0" encoding="UTF-8"?>
<agent-jabber>
    <!--Payload 1-->
    <agent-payload checksum="1234567"
        content-transfer-encoding="base-64"
        content-type="application/x-zip-compressed"
        desc="Description here" filename="Tropical Card.svg"
        filesize="573677"
        timestamp="20040017184631">
        <![CDATA[]]>
        <url>http://server1/Tropical Card.svg</url>
        <url>http://server2/Tropical Card.svg</url>
        <url>http://server3/Tropical Card.svg</url>
    </agent-payload>
    <!--Payload 2-->
    <agent-payload checksum="1111111"
        content-transfer-encoding="base-64"
        content-type="application/x-zip-compressed"
        desc="Description here" filename="Cheshire Cat.svg"
        filesize="102457"
        timestamp="20040017184631">
        <![CDATA[]]>
        <url>http://server1/Cheshire Cat.svg </url>
        <url>http://server2/Cheshire Cat.svg</url>
        <url>http://server3/Cheshire Cat.svg</url>
    </agent-payload>
</agent-jabber>
```

Figure 99. Two files are packaged within the Jabber message.

c. *Message Processing*

The message generation process is done prior to sending. Upon receipt, processes such as archival into database or flat file can be triggered, in addition to invocation of programs to display the binary data; e.g., JPEG and GIF images. This module determines whether the binary data will be embedded in the message. This is based on a predetermined file size (e.g., 1MB). This prevents overloading the Jabber servers and in addition, certain administrators limit the message size. Therefore it is advisable to send small-sized files via the Jabber protocol. The file data is only stored within the CDATA section provided its size is less than the preset limit. Otherwise the CDATA is left empty (See Figure 100), but the header information pertaining to the file, such as file name and size, are kept in the message payload.

```
<?xml version="1.0" encoding="UTF-8"?>
<agent-jabber>
    <agent-payload checksum="1234567"
        content-transfer-encoding="base-64"
        content-type="application/x-zip-compressed"
        desc="Description here" filename="Tropical Card.svg"
        filesize="573677"
        timestamp="20040017184631">
        <![CDATA[]]>
        <url>http://server1/Tropical Card.svg</url>
        <url>C:\temp2\Tropical Card.svg</url>
        <url>ftp://ftpserver3/Tropical Card.svg</url>
    </agent-payload>
</agent-jabber>
```
Figure 100. Binary data, if present, is embedded within the highlighted *CDATA* section.

Upon receipt of a message, this module determines whether it needs to retrieve from binary data from a hyperlink (i.e., if *CDATA* section is empty). Storage locations of the binary data file may reside on a web server (e.g., Apache Tomcat), FTP server or a dedicated agent with web server functionalities built into it. With an Apache web-server, it needs to be administered. On the other hand, having web server functionalities reside in a Jabber-enabled agent reduces the need for additional administration. It may not be advisable, however to burden the agent with additional processes.

Pre-processing activities that are platform or system specific; e.g., image segmentation on UAV imagery, are not included as part of the message processing module. This keeps this module generic and thus extensible to other AUV or non-AUV platforms, as well as improving the performance of the module. Pre-processing modules are responsible for computation of data, representation (e.g., how to capture continuous changing information such as change in water pressure due to an explosion), and generation of results in file or text format.

d. Compression and Decompression

Compression of the message packet to reduce its size is done here. Similarly, decompression is performed at the receiving end. Some file formats are already in compressed form; e.g., GIF, JPEG and PNG do not require further compression. Compress-able file formats include Windows bitmaps and ASCII text. If the zipped form of the file is bigger than the original file size, the original data file is used instead. For this thesis, standard compression and decompression from Java are used. For added security, this module can be easily replaced with a cryptographic module here and on the receiving end.

e. Base-64 Encode and Decode

Binary to text encoding is performed in this module to handle non-textual data; e.g., images, audio, video and compressed data from the compression module.

f. XML Parsing

Since Jabber messages are XML-based, XML parsing and transformation are required. Open-source libraries (Apache Xerces for parsing, Apache Xalan for transformation) are used.

7. Message Generation

The steps to process an outgoing message are given below.

1. Loop through list of files to be sent.
2. For each file, do the following:
    a. Create a <agent-payload> element.
    b. Add file information such as name, size and content-type into the <agent-payload> attributes.
    c. Determine file types based on file extension (e.g., .BMP is 24-bit Windows bitmap and .GIF is Compuserve GIF).
    d. If the files types are GIF, JPEG, EXE, do not compress. Otherwise, perform compression. If compressed data size is greater than original, encode the original data instead.
    e. Check whether the file size exceeds a predefined limit. If no, proceed to compress and encode into CDATA section of message.
    f. If available, always add a list of hyperlinks associated to the file under the <url> tags.
3. Generate XHTML message.
4. Append to the XHTML portion of Jabber message and send it out to designated parties.

The steps to process an incoming message are given below.

1. Loop through list of <agent-payload> elements.
2. For each <agent-payload> element, do the following:
   a. Get file information such as name, size, content-type and content-transfer-encoding.
   b. Check whether there is data in the *CDATA* section of message.
      i. If yes, check type
      ii. Otherwise, loop through the list of <url> tags and try to retrieve the file from one of the storage locations specified within the <url> tag. At the moment, the customize client is able to fetch the file from a web/HTTP server and local/networked file system.
   c. Upon successful retrieval of the file contents, an "AgentPayload" object shall be created.

```
<agent-payload ...
     <url>http://server2/Cheshire Cat.svg</url>
     <url>http://server3/Cheshire Cat.svg</url>
     <url>ftp://www.server3.com/Cheshire Cat.svg</url>
     <url>file://c://server3/Cheshire Cat.svg</url>
     <url>c://server3/Cheshire Cat.svg</url>
</agent-payload>
```

Figure 101. Links to multiple storage locations.

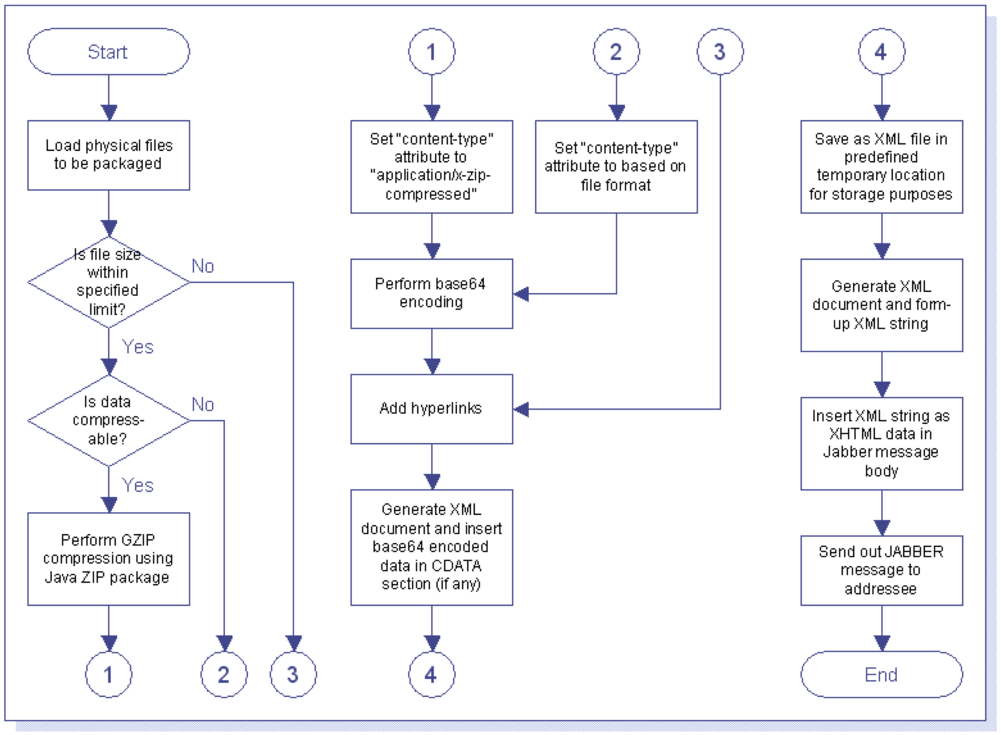

Start
Load physical files to be packaged
Is file size within specified limit?
No
Yes
Is data compress-able?
No
Yes
Perform GZIP compression using Java ZIP package
1
2
3
1
Set "content-type" attribute to "application/x-zip-compressed"
Perform base64 encoding
Add hyperlinks
Generate XML document and insert base64 encoded data in CDATA section (if any)
4
2
Set "content-type" attribute to based on file format
3
4
Save as XML file in predefined temporary location for storage purposes
Generate XML document and form-up XML string
Insert XML string as XHTML data in Jabber message body
Send out JABBER message to addressee
End

Figure 102. Processing of outgoing binary file data before it is sent out via Jabber protocol.

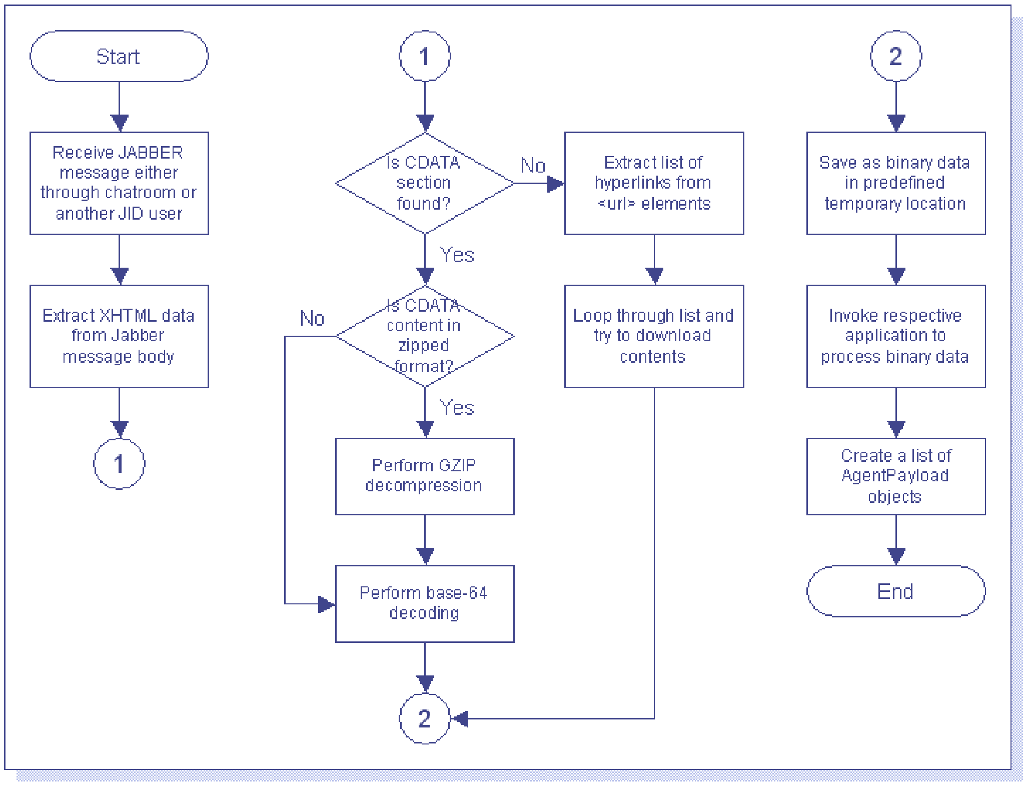

Start
Receive JABBER message either through chatroom or another JID user
Extract XHTML data from Jabber message body
1
1
Is CDATA section found?
No
Yes
Is CDATA content in zipped format?
No
Yes
Perform GZIP decompression
Perform base-64 decoding
2
Extract list of hyperlinks from <url> elements
Loop through list and try to download contents
2
Save as binary data in predefined temporary location
Invoke respective application to process binary data
Create a list of AgentPayload objects
End

Figure 103. Processing of incoming encoded binary data via Jabber protocol.

8. Smack Library

Smack is a library for communicating with XMPP servers to perform instant messaging and chat. The library provides easy machine-to-machine communication and it allows the setting of any number of properties on each message, including properties that are Java objects. The library was developed and maintained by Jive Software, at www.jivesoftware.com (accessed February 2004) and is open-source under the Apache Software License, which allows its incorporation into both commercial and non-commercial applications.

The library is extremely simple to use, yet it has a powerful set of Application Programming Interfaces (APIs). Sending a text message to a user can be accomplished in three lines of code:

```
XMPPConnection connection = new XMPPConnection("surfaris.cs.nps.navy.mil");
connection.login("userA", "passwordA");
connection.createChat("userB@xchat.movesinstitute.org").sendMessage("Hello");
```

Smack provides the *org.jivesoftware.smack* package for the core XMPP protocol, and the *org.jivesoftware.smackx* package for many of the protocol extensions.

G. BENCHMARKS

In the benchmark tests, only the timings for packaging (i.e., compression and base-64 encoding) are captured. Network timings (i.e., Jabber instant messaging) are excluded since the results will be subjected to the network traffic and the available bandwidth. The ASCII plain-text data are from GEOnet Names Server (GNS) at http://earth-info.nima.mil/gns/html (accessed on 15 February 2004). The XML plain-text data were converted from the ASCII data using the GNS class (see Appendix F). A sample procedure to convert GNS plain-text data to XML format is given below.

S/N	Original file size (in bytes)	Encoded file size (in bytes)	Percentage of reduction (in %)	Total time to compress (in msecs)	Total time to base-64 encode (in msecs)	Total time taken (in msecs)
1.	7326956	2380837	67.51	1219	203	1797
2.	5754326	1845969	67.92	1094	250	1657
3.	4378072	395021	90.98	862	440	1642
4.	2106320	652205	69.04	344	141	563
5.	683375	179392	73.75	78	0	156
6.	489429	153828	68.57	63	0	78
7.	402650	98732	75.48	47	0	125
8.	402295	105304	73.82	47	0	63
9.	333117	86264	74.10	110	16	141
10.	243188	78892	67.56	31	94	172
11.	243188	78892	67.56	31	0	63
12.	93972	26147	72.18	16	0	16
13.	63147	19727	68.76	0	0	0

Figure 104. ASCII Plain-text Files achieved on average 72.09% reduction in size.

S/N	Original file size (in bytes)	Encoded file size (in bytes)	Percentage of reduction (in %)	Total time to compress (in msecs)	Total time to base-64 encode (in msecs)	Total time taken (in msecs)
1.	2534082	77473	96.94	240	41	701
2.	737564	31412	95.74	80	40	330
3.	900805	226032	74.91	300	30	591
4.	328028	74560	77.27	90	0	100
5.	46480	5295	88.61	0	0	10

Figure 105. HTML Plain-text Files achieved on average 86.69% reduction in size.

S/N	Original file size (in bytes)	Encoded file size (in bytes)	Percentage of reduction (in %)	Total time to compress (in msecs)	Total time to base-64 encode (in msecs)	Total time taken (in msecs)
1	5941238	867565	85.40	516	172	797
2	1889736	244145	87.08	125	0	203
3	1378706	205665	85.08	78	0	156
4	1120696	147261	86.86	63	0	141
5	1116375	134441	87.96	62	16	156
6	896070	118276	86.80	47	15	156
7	649901	103836	84.02	47	0	63
8	262508	36940	85.93	15	0	31
9	173119	26904	84.46	15	0	15

Figure 106. XML Plain-text Files achieved on average 85.95% reduction in size.

S/N	Original file size (in bytes)	Encoded file size (in bytes)	Percentage of reduction (in %)	Total time to compress (in msecs)	Total time to base-64 encode (in msecs)	Total time taken (in msecs)
1.	3003707	87177	97.10	270	30	590
2.	870106	33792	96.12	80	0	330
3.	891287	224024	74.87	310	20	541
4.	197414	65208	66.97	50	20	80
5.	13134	3039	76.86	0	0	20

Figure 107. X3D Plain-text Files achieved on average 82.38% reduction in size.

S/N	Original file size (in bytes)	Encoded file size (in bytes)	Percentage of reduction (in %)	Total time to compress (in msecs)	Total time to base-64 encode (in msecs)	Total time taken (in msecs)
1.	3003196	86825	97.11	261	30	591
2.	869593	33528	96.14	80	0	330
3.	890645	223820	74.87	300	0	550
4.	174369	64412	63.06	60	20	80
5.	11991	2999	74.99	0	0	20

Figure 108. VRML Plain-text Files achieved on average 81.23% reduction in size.

S/N	Original file size (in bytes)	Encoded file size (in bytes)	Percentage of reduction (in %)	Total time to compress (in msecs)	Total time to base-64 encode (in msecs)	Total time taken (in msecs)
1.	4760390	1687945	64.54	2443	501	3335
2.	3830190	1084981	71.67	1452	410	2163
3.	1300052	417709	67.87	311	40	571
4.	854570	171312	79.95	141	30	211
5.	361041	149080	58.70	170	10	200
6.	220866	92392	58.17	70	10	100

Figure 109. SVG Plain-text Files achieved on average 66.82% reduction in size.

H. SUMMARY

This chapter has proposed a solution to package text and binary data to be sent via Jabber instant messaging protocol. Compression of data was done using standard Java classes. The Java classes compressed both text-based (including XML-based file) and binary files such as images. In general the percentage of compression achieved is 75% of the original file, higher for text-based files. As for Windows bitmap images, the compression ratio depends on the type of image that has been stored. Therefore a better way to ensure optimal compression is to convert the Windows bitmap images to PNG or JPEG format on the fly. Of note, JPEG is lossy and may cause degradation in image quality. Therefore PNG format is favored for its lossless' nature. To achieve better compression ratio for XML data, XML Schema-based Binary Compression (XSBC) (Serin2003) can be used to complement the Java classes. XML files are handled by XSBC, whereas non-XML files such as plain-text or images are handled by the proposed mechanism. Forward Error Correction can be introduced to ensure reliability of data transmission and receipt over noisy communication channels.

On average, the base-64 encoding of the binary data to text data, for storage in the CDATA section of the Jabber XML-based message, increased the file size by 33%. A better binary to text encoding scheme (e.g., using Huffman algorithm), can be pursued in future work.

It is important to note that besides size, the time taken to run the compression process also plays an important part. An algorithm may be good at generating a smaller sized file, but the time taken may be so long that it is unacceptable to the user.

There is a physical message size limit set on the Jabber server. This is to prevent overloading the Jabber server. Although the proposed mechanism introduces the use of hyperlinks to circumvent the potential issues, a more robust solution should be pursued; e.g., if the size is too big, the file is automatically posted on a web server using standard HTTP POST mechanism.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TASK COLLABORATION USING AGENTS

A. INTRODUCTION

Multiple vehicles operating in a coordinated manner can be more effective than a single one. For instance, a cluster of coordinated Autonomous Underwater Vehicles (AUVs) can search a coastal area for mines more effectively than a single vehicle, however handling unanticipated events (novel or completely unexpected) is difficult, since they are sometimes hard to detect, much less diagnose and respond to, even for a single vehicle. If the AUV is part of a cooperative or collaborative distributed multi-agent system, the problem is compounded. The AUV controller must now be concerned about what the event means for the others, the group as a whole, and their shared mission/goal. Multi-agent event handling is complicated by uncertainty and lack of knowledge about other agents' intentions/goals; it is exacerbated by the low bandwidth of communication channels available for use in the ocean.

An agent also needs to be able to communicate with other agents to fill in the gaps in its models or hypotheses, to establish mutual beliefs and confirm expectations, and to negotiate responsibility for the different tasks during event handling. Some key challenges include:

a. To determine an efficient way to deploy multiple Autonomous Underwater Vehicles (AUV) for collaborative work such as mine counter-measures missions.

b. To determine what is the optimal number of AUVs to be used for a scenario (e.g., based of time of completion).

c. To investigate amount of deviation between real-world dynamics and simulated ones in a virtual environment.

d. Ability to play out more scenarios at less cost compared to live sea-trials.

This chapter discusses the potential use and design of AUV agents.

Following the MAS = {Environment, Objects, Agents, Relationships, Operations, Laws} approach [Ferber 1999], the various components of the system are discussed in the following sections.

B. ENVIRONMENT

The AUVs operates in the ocean (sub-surface). The area of operation is determined by the physical constraints of the AUV, as depicted below:

a. Endurance. The fuel tank size of the AUV is fixed.

b. Speed. The maximum speed of the AUV is fixed.

c. Sensor.

d. Communications.

For purposes of simulation, there is a finite number of AUVs, maximum five. Similarly the maximum number of Communications Stations (include land, air and sea-based) is set to five. The need to limit the number of AUVs and Communications Stations is to better model the real-world environment, where it is impossible to have an infinite number of resources. For the experimental runs, it is possible that zero or more obstacles may be placed in the environment.

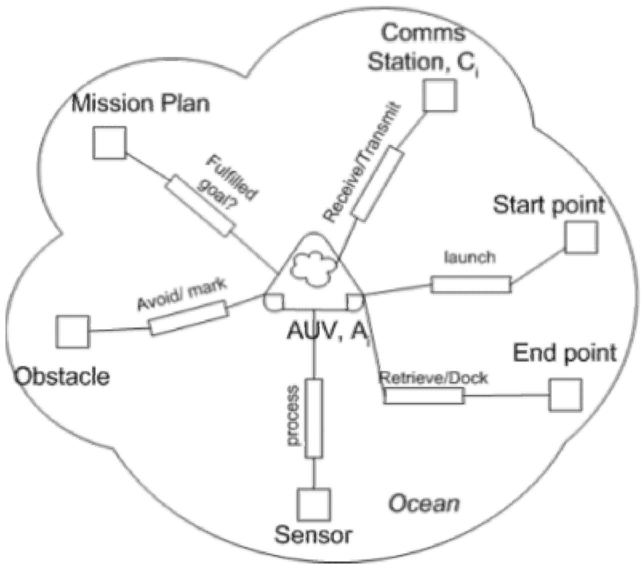


Figure 110. Agent Boundary.

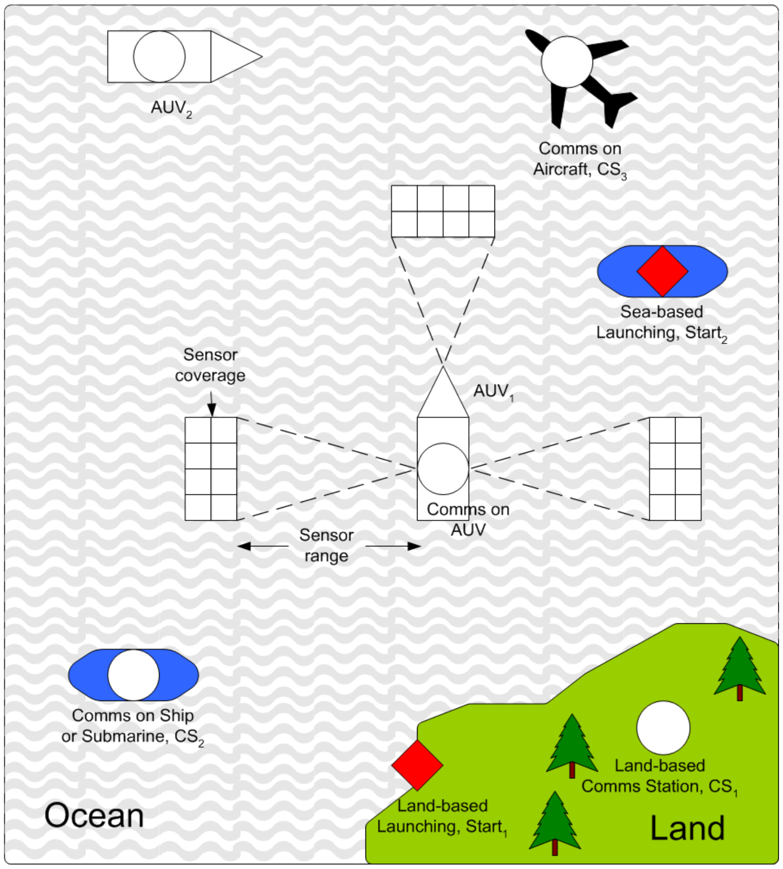

AUV$_2$
Comms on Aircraft, CS$_3$
Sea-based Launching, Start$_2$
Sensor coverage
AUV$_1$
Comms on AUV
Sensor range
Comms on Ship or Submarine, CS$_2$
Ocean
Land-based Comms Station, CS$_1$
Land-based Launching, Start$_1$
Land

Figure 111. AUV operating environment.

C. OBJECTS

List of objects and their attributes:

1. AUV

S/N	Attribute	Description
a.	ID (unique numeric value)	This serves as an identifier for each AUV. The identifier is used to distinguish between multiple AUV agents in a networked environment. It is generated programmatically by the AUV Workbench at runtime.
b.	Position (numerical value in meters)	Indicates the location of the AUV in 3D space; i.e., X, Y and Z coordinates.
c.	Orientation (numeric value in degrees)	Indicates the row, pitch and yaw of the AUV.
d.	Heading (numeric value in degrees)	Determines which direction the AUV is heading.
e.	Speed (numeric value in knots)	Speed of the AUV.
f.	Endurance	Based on fuel consumption and the time it has been operating.

S/N	Attribute	Description
g.	Type of Sensor	Type of sensor available on-board the AUV.
h.	List of Comms Stations	Keep a list of communications stations.
i.	List of Obstacles	Historical list of obstacles encountered.
j.	Physical	Physical dimensions of the AUV, including maneuverability limitations, which constraint the maximum turning radius of the vehicle.
k.	Status	A list of possible status: “Ready”, “Damaged”, “On-shore”, “Deployed” and “At Comms Station”.

Table 29. AUV Attributes.

2. Sensor

S/N	Attribute	Description
a.	Name (string value)	Description of sensor.
b.	Category (string value)	Which category does the sensor belongs to; e.g., sonar or optical (static picture or motion video)?
c.	Range (numeric value in meters)	Which category does the sensor belongs to; e.g., sonar?
d.	Status	A list of possible status: “Ready”, “Damaged”, “On-shore”, “Deployed” and “At Comms Station”.
e.	Footprint (numeric value in area per square meters)	What is the coverage of the sensor?
f.	Position of AUV (enumeration)	Left, right, up or down. Therefore an AUV can have 4 sensors mounted.
g.	Readiness State (Boolean value)	Up or down.

Table 30. Sensor Attributes.

3. Communications Station

S/N	Attribute	Description
a.	Name (string value)	Description of comms station.
b.	Category (string value)	Type of station; e.g., ship-based, land, aircraft or sub-surfaced vehicle. Also include whether it is stationary or moving.
c.	Position (numerical value in meters)	Indicates the location of the Comms Station in 3D space i.e., X, Y and Z coordinates.

Table 31. Communications Station Attributes.

4. Obstacle

S/N	Attribute	Description
a.	Name (string value)	Description of obstacle.
b.	Category (string value)	Type of obstacle (e.g., mines and marine life such as fish and kelp).
c.	Position (numerical value in meters)	Indicates the location of the position in 3D space as detected by the AUV.

Table 32. Obstacle Attributes.

5. Mission Plan

To be loaded into AUVs. Mission plans are also used to define the initial goal of the AUVs.

S/N	Attribute	Description
a.	Name (string value)	Description of Mission Plan for identification purposes.
b.	AUV specific information.	Initial position and orientation of AUV.
c.	List of waypoints	Positional check-points for the AUV to be at, for a given time.
d.	Start/End points	AUV launch position and retrieval/docking positions.

Table 33. Mission Plan Attributes.

6. Launching/Pick-up Point

The position where the AUV is launched and picked up upon completion of goals or when out of fuel/time.

D. AGENTS AND ACTORS

The presence of multiple agents impact event handling in several ways. First, an agent may notice an event that does not directly concern it, but impacts another agent. In this case, the detecting agent can consider notifying the other agent about it. Second, there is the possibility that multiple agents detect the same event simultaneously. If this happens, the agents must coordinate their event-handling activities to avoid confusion and counter-productive work. Third, other agents can serve as a source of information (solicited or otherwise). Some agents may be in a better position with respect to the knowledge they have or can easily obtain to do diagnosis or important assessment, and others may be better situated to carry out the responses. At the very least, a detecting agent notifies the affected agents of actions it is taking, whether in service, diagnosis, or in response to an event. Figure 112 illustrates the relationships between inputs/outputs and the agent interior.

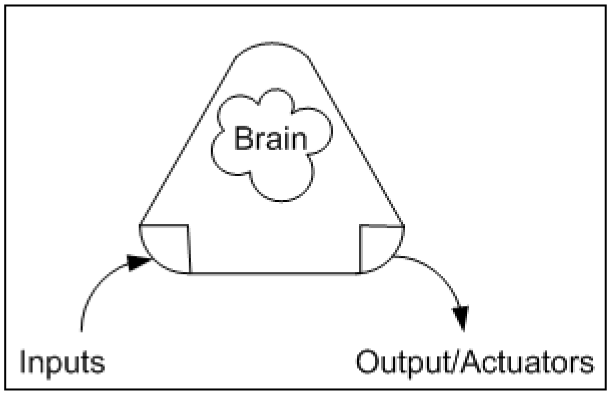


Figure 112. Agent Overview.

Input Suite	Output/Actuator Suite
Mission Plan.	Percentage of completion of Mission Plan.
Initial position, orientation and speed.	Move the AUV around the environment.
Sensor inputs.	AUV attributes.
Inputs from Communications Stations; e.g., to change target priority/position.	All attributes can be accessed via “setter” and “getter” methods; e.g., “getCurrentPosition()”
Unique ID.	To uniquely identify an AUV agent.

Table 34. Agent input and actuator suite.

Each AUV agent has the following attributes/states stored in the “brain” of the agent. The agent interior is hidden from other agents unless the information is exposed via the agent’s output or actuator suite.

- Endurance. Elapsed time is computed at the start of the experiment based on the amount of fuel available.
- List of obstacles encountered. Obstacles include mines, marine life and other AUVs.
- Mission scripts of AUV commands. List of waypoints and the corresponding arrival/departure time at each waypoint. To improve

interaction between both AUVWorkbench and other applications, the mission scripts/commands are defined in XML format (see sample Mission Script below).

```
<?xml version="1.0" encoding="UTF-8"?>
<AUVMissions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:/AUVWorkbench/bin/scripts/missionScripts/AUVMiss
ion.xsd">
    <Mission>
        <Profile/>
        <Commands>
            <Position course="0" depth="5" portPropSpeed="27"
                standoff="2" starboardPropSpeed="26" thruster="on"
                timeout="50" x="12" y="55"/>
            <Waypoint course="180" depth="15" portPropSpeed="27"
                standoff="" starboardPropSpeed="26" thruster="on"
                timeout="5" x="95" y="55"/>
            <Waypoint course="180" depth="15" portPropSpeed="27"
                standoff="" starboardPropSpeed="26" thruster="on"
                timeout="5" x="122" y="72"/>
            <Waypoint course="270" depth="5" portPropSpeed="27"
                standoff="" starboardPropSpeed="26" thruster="on"
                timeout="5" x="68" y="64"/>
            <Hover course="270" depth="5" portPropSpeed="27" standoff=""
                starboardPropSpeed="26" thruster="on" timeout="5" x="12" y="70"/>
            <Speed speed="0"/>
            <Thruster enabled="false"/>
        </Commands>
    </Mission>
</AUVMissions>
```

S/N	Element	Description	Measurement Unit
1.	AUVMissions	Root element.	-
2.	Mission	There can be many “Mission” elements in the same Mission script.	-
3.	Profile	To be used to define area of interest/operations.	-
4	Commands	List of commands to be sent to the AUV.	-
5.	Position, Waypoint, Hover	Positional data.	-
6.	Speed	Speed of both port and starboard thrusters.	Revolutions per minute.
7.	Thruster	Rear thrusters.	Boolean.
8.	Timeout	Time period to wait.	In seconds

Table 35. Mission script XML tag set.

The same mission plan is loaded into all the AUVs. The course of action is negotiated between the AUV agents. Each AUV is able to determine its location based on its heading, speed and time elapsed (with reference to the start point). When a Communications Station is within the AUV's transmission range, the agent polls the station for its position. Using the coordinates returned by the Communications Station and distance between the Communications Station and AUV, the agent tries to check whether its internally computed position is correct. Here is how the goals are defined:

S/N	Goal	Description/ Measurement Method	Course of Action
1.	Movement from point-to-point	Moving to a specified position (defined in the Mission Plan) within stipulated time (with a user-definable tolerance; e.g., 15 minutes).	Update "WAYPOINT REACHED" status in Mission Plan. It tries to update the Comms Station or any AUV within its vicinity.
2.	Movement within Area of Interest	Move into a predefined region of operations.	-
3.	Movement by following a leader	2 possible scenarios: • Right from the start, follow a pre-determined leader. • Upon receipt of indication that a detection has occurred.	-
4.	Object/Obstacle Detection	Based on sensor input, the agent is able to know the general location of the obstacle and whether it is moving or stationary. It is possible for an obstacle to be larger than the sensor's field of view.	Slow down and try to identify the obstacle (a mine, another AUV or fish?). If there is a high probability that it is a mine, it tries to get another AUV to double-check. And at the same time, it tries to inform the Comms Station.
5.	Object/Obstacle Identification	Detection is normally followed by identification and confirmation of target objects.	-
6.	Collision	Occurs when the AUV is caught "off-guard" due to limited sensing capability (note: the sensors on-board do not allow the AUV to have full sensing of its neighboring environment).	Locate other AUVs that are close-by. Locate Comms Stations that are close-by. Surface and/or dock if necessary.

S/N	Goal	Description/ Measurement Method	Course of Action
7.	Position Update	When it is within range of another AUV or Comms Station, it checks whether the position it has computed is correct.	If position is wrong (outside a Tolerable limit; e.g., 25m), it makes adjustment and tries to move to the desired position. Once at the correct position or along the way, it checks whether the list of waypoints it had supposedly completed is correct (within a tolerable limit; e.g., 25m).
8.	Endurance	Based on the time remaining, it extrapolates whether there is enough time to complete its task.	Speed up or slow down so that all its targets/waypoints can be achieved.
9.	Communications	Transmission of data back to comms stations.	-

Table 36. Agent goal definition.

For each of the goals defined, there is a priority (similar to "traffic-light" system) assigned to them. The priority shall have three states – low, medium and high.

E. RELATIONSHIPS

1. Define initial goals and update status as required (Mission Plan ↔ AUV).
2. Encounter or detect an obstacle (Obstacle → AUV).
3. Sensor Input (Sensor → AUV).
4. Transmit findings to Communications Stations or receive positional data/new orders (Communication Stations ↔ AUV).

F. PROCESSES AND OPERATIONS

1. Follow waypoints defined in Mission Plan.
2. Compute its position internally based on its speed and elapsed time.
3. Update "waypoint reached" status upon arriving or bypassing a waypoint.
4. Upon obstacle detection or encounter, look for closest AUV agent and/or Communications Stations to share information.

5. When within range of another AUVs, share the following information; 1) own percentage of Mission Plan completion; 2) own position.

6 When within range of Communications Station, the AUV checks whether its own computed position is correct by referencing the Communications Station's position (pre-loaded into AUV). If not, perform compensatory action. In addition, it checks whether the list of waypoints that it thought it passed is correct.

G. SUMMARY OF LAWS

1. Communications between AUV-AUV and AUV-Communication Stations are possible only when within range. This is a "*Many-to-Many*" relationship.

2. Communications shall be initiated by the AUV or Communications Stations.

3. Communications Stations are manned or remotely operated by humans. Therefore their locations are always precise since additional equipment is available to geo-reference them (GPS-enabled). All Communications Stations' coordinates are pre-loaded or made known to AUV agents.

4. To simplify the system, there are two types of sensors, namely sonar and optics. Both types of sensors shall operate ideally with the given range and coverage. In the real-world, these sensors rarely operate to their optimal performance due to oceanic conditions.

5. The launching point and the retrieval point of the AUV may differ.

6 The team of vehicles is moving in an environment of known dimensions, searching for target of interest. The vehicles are assumed to be equipped with: 1) target sensing capabilities for obtaining a limited view of the environment; 2) wireless communication capabilities for exchanging information and cooperating with one another; and 3) computing capabilities for processing the incoming sensor data and making dynamic guidance decisions.

7. For each AUV agent, there is only one Mission Plan.

8. Fuel consumption by the AUV is constant; therefore the AUV is able to operate for a fixed time period; e.g., 3 hours.

9. The agents are equipped with sensors to view a limited region of the environment they are visiting, and are able to communicate with one another to enable cooperation. The agents are assumed to have some "physical" limitations including maneuverability limitations, fuel/time constraints and sensor range and accuracy.

H. AGENT IMPLEMENTATION

1. Concept of Connector-Ticket Pair

Packaging and Tagging of Raw Inputs. Raw inputs from sensors (e.g., sonar or video feed) are first packaged. Packaging involves formatting the data into an agent-readable form (e.g., following a particular XML schema or template). This is followed by tagging. The tagging process is to add connectors and/or tickets to the packaged data. The connector-ticket pairs allow the data to interact with the agent's set of connector-ticket pair of goals. Finally the tagged data is passed into agent's space for interaction. Integration networks (IN) are formed as related (in terms of time and event) tagged data is grouped together. In addition, if double-scope blending takes place, new generic spaces can be created [Turner 2002]. With the creation of new generic spaces, the agent is in fact shaping its perception of the environment.

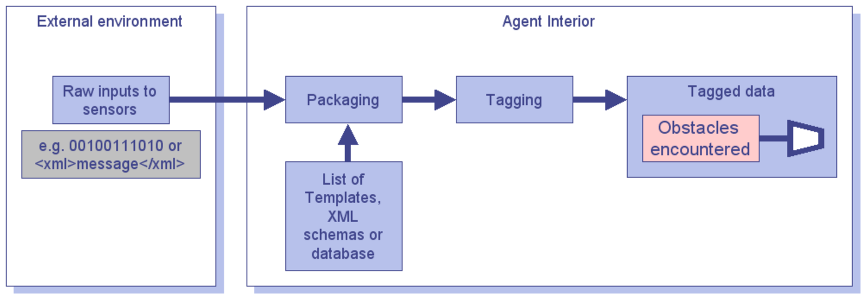


Figure 113. Connector-Ticket - Packaging and Tagging.

Matching tagged data. At the end of each tagged data, there are either one or many connectors that match the ones that are extended from the agent's goals (See Figure 108). Tagged data and connectors form a ticket. When there is a match, the agent may choose to retract the fulfilled goal or trigger a new set of goals. At the moment, these actions are defined as a template within the agent. With better understanding of the conceptual blending principles, it may be possible to get the agent to formulate new goals that will aid in its fulfillment of goals/functions.

Below is a possible scenario on how goals can be altered based on the agent's perception of the environment and its interaction with other agents through the use of connectors and tickets.

At the start of the run, agents are given basic goals; e.g., "Move along a predefined path or an area of interest within a given time". If the AUV sensors detect an object, the basic goals are either upgraded or replaced (i.e., retracted) by more complex ones. The complex goals may vary from getting another AUV to perform identification or confirmation of contact, or to track the object/obstacle for a predefined time period, or surface and transmit data back to communications stations.

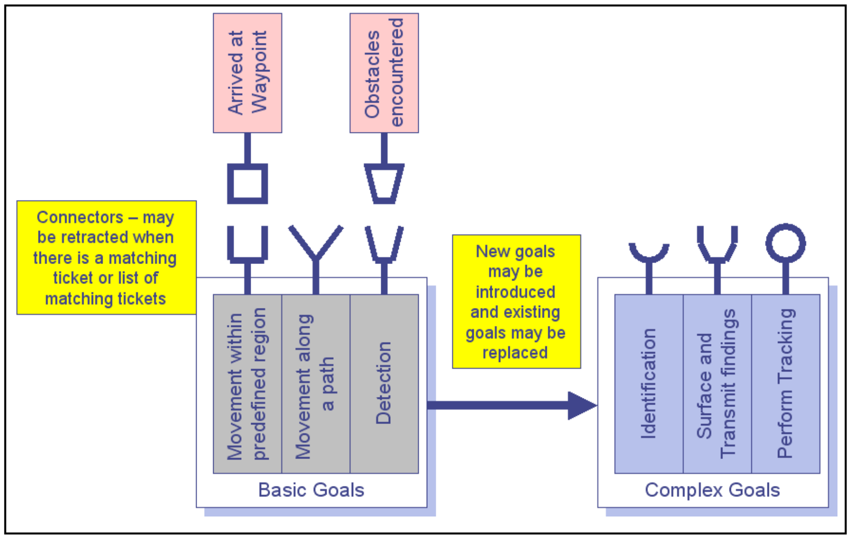

Arrived at Waypoint
Obstacles encountered
Connectors – may be retracted when there is a matching ticket or list of matching tickets
New goals may be introduced and existing goals may be replaced
Movement within predefined region
Movement along a path
Detection
Basic Goals
Identification
Surface and Transmit findings
Perform Tracking
Complex Goals

Figure 114. Connector-Ticket Matching.

I. AGENT-TO-AGENT COMMUNICATIONS

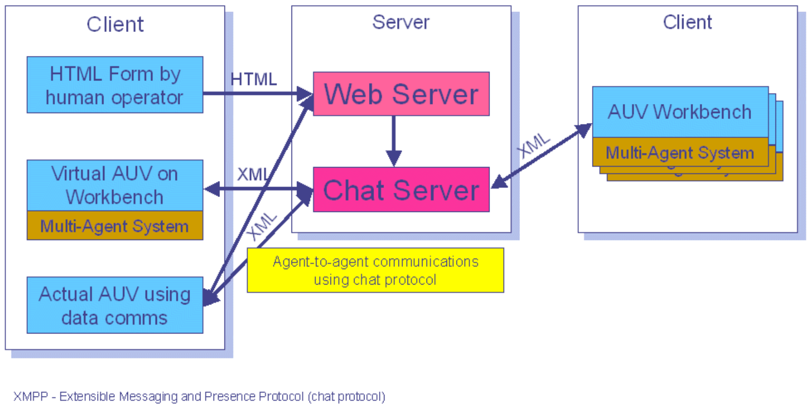
1. Agent Identifier

The system makes use of the chat/Jabber ID (e.g., *XTCServlet@xchat.MovesInstitute.org*) to distinguish between multiple agents across a list of networked AUV Workbench applications.

For each workbench, there is one agent identifier. The user is able to change the agent's identifier from within the workbench. Note: the workbench must have connectivity to a chat/Jabber server.

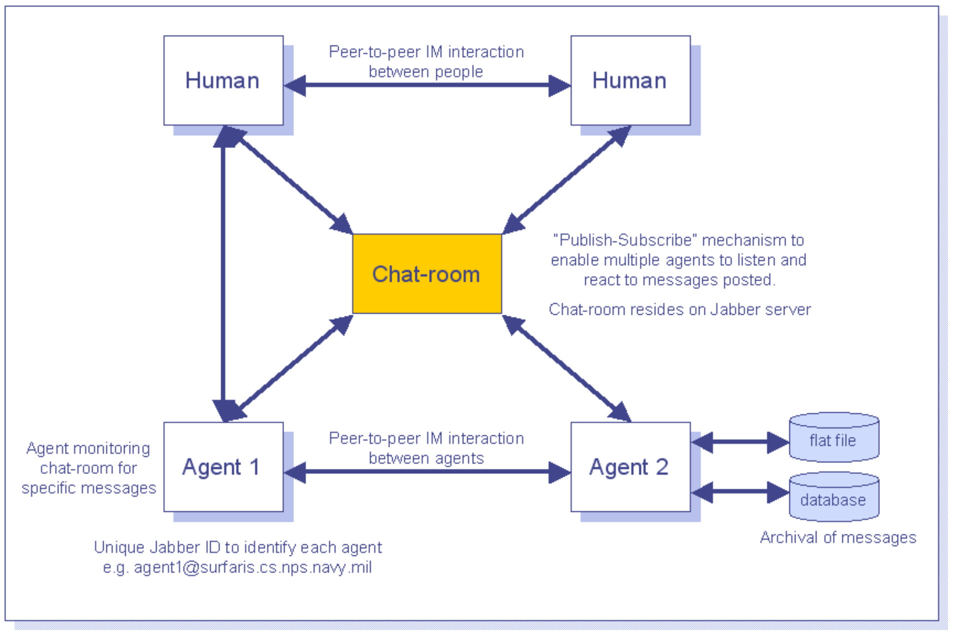
2. Communications

Extensible Messaging and Presence Protocol (XMPP) is used as the means for communications between agents. The techniques for packaging and transporting agent messages via XMPP are the same as those discussed in Chapter IV.



Client
HTML Form by human operator
Virtual AUV on Workbench
Multi-Agent System
Actual AUV using data comms
HTML
XML
XML
Server
Web Server
Chat Server
Agent-to-agent communications using chat protocol
XML
Client
AUV Workbench
Multi-Agent System
XMPP - Extensible Messaging and Presence Protocol (chat protocol)

Figure 115. Agent-to-agent communications using XMPP.



Human
Peer-to-peer IM interaction between people
Human
Chat-room
"Publish-Subscribe" mechanism to enable multiple agents to listen and react to messages posted.
Chat-room resides on Jabber server
Agent monitoring chat-room for specific messages
Agent 1
Peer-to-peer IM interaction between agents
Agent 2
flat file
database
Archival of messages
Unique Jabber ID to identify each agent e.g. agent1@surfaris.cs.nps.navy.mil

Figure 116. Human and Agent interaction via Jabber chat room.

3. Strategy for Data Collection

- Compute number of AUVs and Communications Stations available.
- Compute the percentage of completion of Mission Plan.
- Keep track of obstacles encountered (mines and AUVs).

4. Data Analysis

a. Initialization Phase

The system randomly assigns a unique ID for each of the AUV agent. The mission plan is loaded into the AUVs together with start and end locations, and Communications Station locations.

Once the Mission Plan has been loaded, the agent computes the feasibility of completing the plan, within the required time allowed (or fuel constraint). If not, it informs the user to make changes to the Mission Plan. If the plan is feasible, the agent generates an internal "Search Map". Unvisited waypoints (in RED, Figure 117) are marked. Similarly for Start/End points (in GRAY, see Figure 117) and the locations of Communications Stations (in CYAN, see Figure 117).

b. Start/During the Run (AUV Execution)

- When a *waypoint* is reached, it is marked (in GREEN, Figure 117). The time is recorded and compare against the time specified in the Mission Plan. A tolerance of 5-10 minutes is allowed. If the time discrepancy is too great, the agent tries to compensate by increasing speed to the next waypoints.
- When the agent senses an *obstacle*, it slows down and tries to identify what it is (e.g., by shape and size). To identify the obstacle, it instructs the vehicle to move around the obstacle to gather more data. If a particular type of obstacle (e.g., mine) has been found, it tries to inform the Communications Stations and other AUVs. And if neither an AUV nor Communications Station is within its vicinity, it proceeds to the closest Communications Stations and/or moves towards a previously known AUV's direction. The obstacle is marked in the agent's internal "Search Map". Additional information on the obstacle such as size, moving/stationary and type are also captured in the corresponding grid square in the "Search Map".

- When in contact with another *AUV* or within the transmission range of *Communications Station*, it shares its position, orientation, heading and speed with it. Similarly the other AUV shares the same attributes.

c. Possible Strategies

- *Follow the path where there is minimum overlap with other agents*. Since the agents are able to share their new information about the search region, it is natural that they may select the same search path as other agents (especially since in general they will be utilizing the same search algorithm). This will be more pronounced if two agents happen to be close to each other. However, in order to minimize the global uncertainty associated with the emergent knowledge of all agents, it is crucial that there is minimum overlap in their search efforts. This can be achieved by including a cost function component that penalizes agents being close to each other and heading in the same direction. This component of the cost function can be derived based on the relative locations and heading direction (angle) between pairs of agents. [Polycarpou 2001]
- *Follow the path that maximizes coverage of the highest priority targets*. In mission applications where the agents have a target search map with priorities assigned to detected targets, it is possible to combine the search of new targets with coverage of discovered targets by including a cost component that steers the agent towards covering high priority targets. The cost component is based on the target's characteristics such as shape, size, mobility and its overall effect on the mission (e.g. mines have a higher priority). This leads to a coordinated search where both coverage and priorities are objectives.
- *Follow the path toward highest priority targets with most certainty*. In some applications, the energy of the agent is limited.

In such cases it is important to monitor the remaining fuel and possibly switch goals if the fuel becomes too low. For example, in search-and-engage operations, the agent may decide to abort search objectives and head towards engaging high priority targets if the remaining fuel is low. Environment factors such as sea state will also affect the operational effectiveness of the AUV sensors as well as its maneuverability.

- *Follow the path toward targets where there will be minimum overlap with other agents*. Cooperation between agents is a key issue not only in search patterns but also in engagement patterns. If an agent decides to engage a known target, there needs to be some cooperation such that no other agent tries to go after the same target; i.e., a cooperative engagement strategy is utilized. On the other hand, this strategy will depend on the availability of the weapon systems onboard the AUV. Multiple attacks will increase the cumulative probability of kill, but this has to be weighted against the probability of not having resources to search for other targets or even to react in time to another target once the AUVs are on-route to the first known target. At the same time, there is also a probability of losing the first target and reacquisition will be required before engagement can commence.

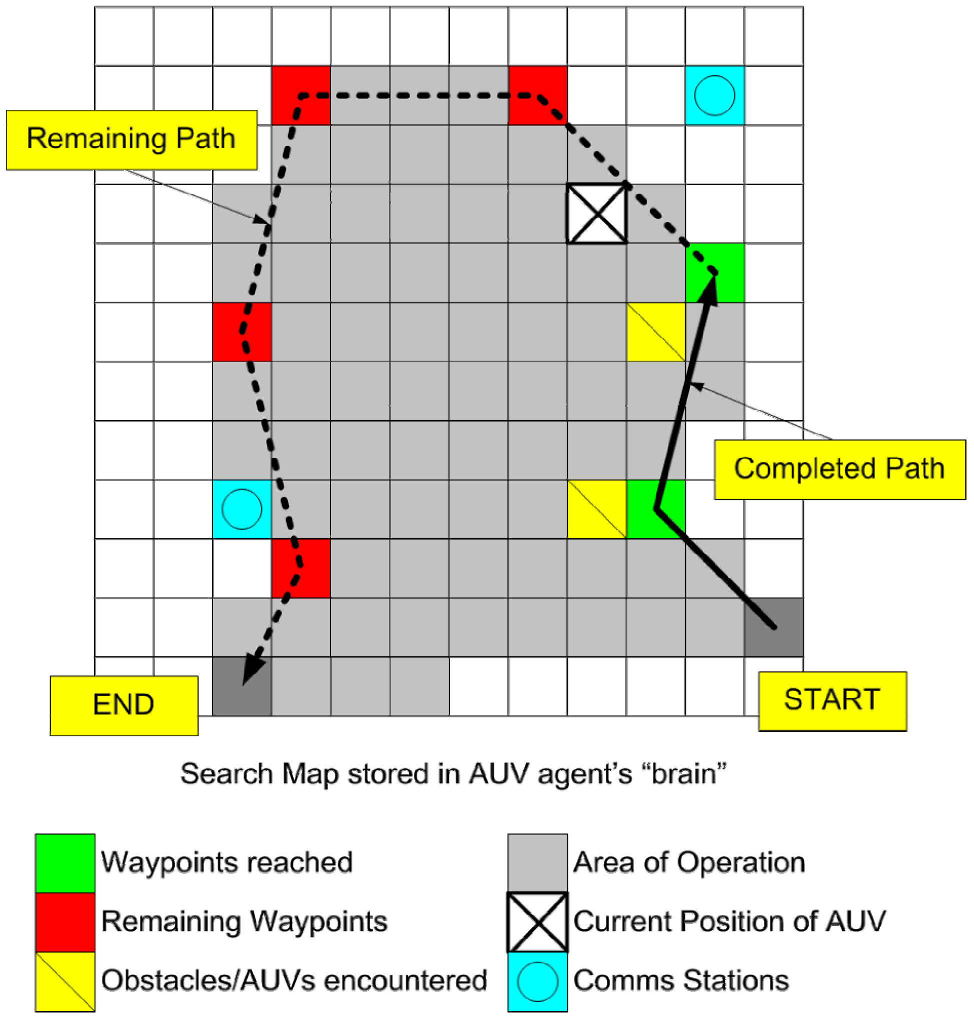


Figure 117. AUV Agent - Search Map.

J. SUMMARY

This chapter discusses the design of the AUV agent attributes and possible strategies to define the agents' goals. Means of communications between agents using Jabber instant messaging were presented. An example application to construct a search map is described.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

The main purpose of this work was to design and implement a common platform for AUV mission planning and analysis. The end product is the AUV Workbench. Using Java-based open-source libraries for functionality, Extensible Markup Language (XML) for data storage and exchange, and a component-based framework, the AUV Workbench provides an intuitive cross-platform-capable tool with extensibility to provide for future enhancements such as agent-based control, asynchronous reporting and communication, and loss-free message compression.

In addition, this thesis has explained the suitability of Jabber instant messaging for text and file messaging in a tactical environment. Exemplars have shown that the XML backbone of this open-standards technology can be leveraged to enable both human and agent messaging, providing powerful improvements over current systems.

B. RECOMMENDED FUTURE WORK

1. Overview

This thesis established the foundation for future work for modeling and simulation of AUVs. This work has demonstrated that the AUV Workbench provides a test-bed for emergent AUV technologies and can assist in the development of traditional and agent-based methodologies. Additionally, the flexible design of the Workbench facilitates potential extensions to serve operational needs. A list of recommendations is given in this section.

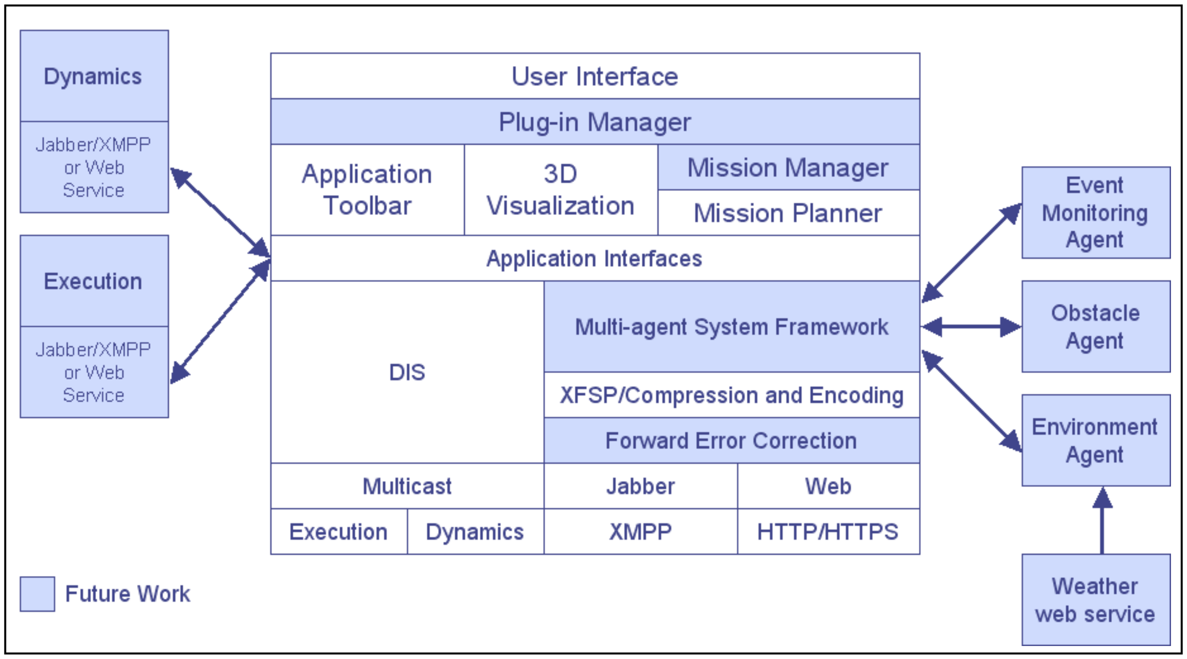


Figure 118. Modular overview of future work.

2. AUV Multi-Agent System Framework

Most AUV missions neither require nor permit constant human oversight. Operating conditions, adverse environmental conditions or inherent limitations of underwater communications paths can cut off communications with the vehicles. For example, a covert surveillance or reconnaissance mission precludes all but the most minimal communication with the vehicle. Therefore if a virtual AUV agent is able to simulate the real AUV in water, it is able to provide human operators with a visual or audible cue on its whereabouts. When the real AUV surfaces, the virtual AUV synchronizes its position and status (e.g., sensor data or equipment failure). While the actual AUV is in water, the human operator may re-task the AUV using AUV agent, with the re-tasked orders transmitted to the actual AUV.

Ideally, AUVs are capable of acting truly autonomously for long periods of time in challenging, unpredictable environments. As the missions undertaken by the AUV become more complex, it becomes difficult for the human to keep up, making agents potentially useful. A set of rules is given to the agents. The human operator intervenes when there is a conflict or when a critical condition arises (i.e., system failure or mine detection).

Just as the XML-based mission script provides low-level commands to the AUV, the same mission script can be extended to define goals in an agent system. This is similar to strategic level commands. (Duane 1996) The agents can be developed using Connector-based MultiAgent System (CMAS) library. (Hiles2004)

AUV reactions are based on its onboard suite of sensors (e.g., when a mine is encountered, loiter to verify, or surface to report). Picture the following: an AUV is in water, a human controller on ship or shore running the AUV Workbench, introduces (drag-and-drop) a virtual obstacle into the virtual environment; a daemon agent pushes this piece of information out to the real AUV's sensors to "simulate" the detection of an obstacle; the AUV in water reacts accordingly.

Instead of having cardboard enemies, we have a more realistic agent that models the enemy. This would help blur the line between simulation and real world operations and give a "practical" use of simulations. Of note, there are still issues such as bandwidth and latency with AUV communications that needs to be solved. Bolder AUV deployment concepts can be tried out; e.g., perform dynamic re-tasking once the AUV is in the water. The agent system provides data to support or confute. At the same time, the system provides details on what is the "cost" involved (in terms of potential loss of target and its endurance) when an AUV is directed to another supposedly higher priority target.

3. Development of Collaborative Sensing Strategy Using Dissimilar AUVs

The current robot execution module is based on the NPS ARIES AUV. This software can be replaced by another AUV from an industry partner or academic institution. Once implemented, NPS will have a wide variety of AUVs to try out different scenarios with dissimilar AUVs, instead of just the NPS-specific vehicles. Picture a tactical scenario whereby a planner defines the area of operations, its conditions and constraints. From a library of AUV models (which includes the virtual 3D representation and the robot software), the system comes out with a list of recommendations. The list may comprise dissimilar AUVs. The most important point is not who's AUV is better, but how can the mission best be accomplished.

The posting of the AUV Workbench application online as open source with executeable binaries makes this vision more realizable, now that developers and partners from other research institutions or industry, can download the application and work on it.

4. Simulation of Targets/Obstacles

The topic of AUV obstacle avoidance is a well-researched area. The challenge is to develop an obstacle avoidance agent that runs alongside the Workbench. At the same time, it is necessary to develop an obstacle generation or simulation module. The obstacle-simulation module dynamically introduces obstacles such as marine life or mines into the virtual environment. At the moment, under the Event Monitoring module, obstacles are displayed as targets. These targets contain only static information and are not “live”. This will cause issues when multiple sightings of the same target (e.g., mine) are reported. These will be plotted multiple times in the Workbench. A better approach is to make “live” targets, i.e., link them to a centralized agent that is responsible for the tracking and aggregation of data pertaining to targets. There can be an agent for each type of obstacles or targets, one for mines and another for ships. A target representation language using XML can be defined. A mine target may look like the one below:

```
<?xml version="1.0" encoding="utf-8"?>
<Target category="mine" type="subsurface"
        classification="unknown" reportedBy="AUV1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\auv\Workbench\Targets\auvTargetLangu
age.xsd">
      <Report dateTime="20040101103000">
            <Position x="100" y="100" depth="15"/>
            <Size length="1" breadth="1" height="1"/>
      </Report>
      <Report dateTime="20040101104500">
            <Position x="100" y="103" depth="13"/>
            <Size length="1" breadth="1" height="1"/>
      </Report>
      <Report dateTime="20040101104500">
            <Position x="100" y="103" depth="13"/>
            <Size length="1" breadth="1" height="1"/>
      </Report>
     <Details desc="More details here">
            <Url>http://tacticalsvr/sightingMine.htm</Url>
            <Url>http://tacticalsvr/video.jpg</Url>
     </Details>
</Target>
```

Figure 119. Proposed XML-based representation of Mine Target.

5. Simulation of Environmental Conditions

The real world conditions are more dynamic and unpredictable. An environment agent is responsible to feed environmental data such as sea-state and ocean current to the Workbench. To lend more realism to the virtual environment, a web service that subscribes to real weather data can be developed. A potential sub-module is to simulate interference and unreliability in communications due to weather conditions.

6. Plug-in Framework

To facilitate further development and use of the Workbench, an important addition is to have a robust component plug-in framework. This framework would consist of a plug-in manager that allows the user to add, remove or configure plug-in modules. This approach is more flexible and robust compared to compiling everything into the Workbench, which may or may not get used. The set of rules and configurations are defined in an XML schema. A sample representation is given below:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Class name="Plugin" returnType="String">
      <Method name="display" returnType="void">
            <Parameter type="int" default=""/>
            <Parameter type="String" default="Description"/>
      </Method>
      <Argument value="mine.xml"/>
      <Argument value="mine.gif"/>
</Class>
```

Figure 120. XML-based representation of Plug-in Class

One approach to develop the plug-ins framework is the use of Java Reflections whereby introspection of classes (plug-in module) can be performed at runtime. Reflection enables Java code to discover information about the fields, methods and constructors of loaded classes, and to use reflected fields, methods, and constructors to operate on their underlying counterparts on objects, within security restrictions. This is an advanced topic in Java programming. Another possible approach is to use Jabber instant messaging for the communications between the plug-in modules and the Workbench application. Again, the messages are defined in an XML schema. The Jabber instant messaging solution developed in this thesis will be useful as it handles data as well as binary files.

7. AUV Mission Manager

An AUV Mission Manager should be introduced to handle the definition and execution of multiple AUV mission plans. The Mission Manager is responsible for the invocation and message passing between the various AUVs in the 3D display and agent environment.

A good approach to implementing the Mission Manager is to have decentralized "Execution" and "Dynamics" processes. These processes might be written as web services or agents using the Jabber protocol for data and file message exchange.

8. User Interface Enhancements

a. Manipulate Multiple Missions in 2D Mission Planner

At the moment, the underlying software architecture supports multiple missions, but the 2D Mission Planner graphics display does not. A mission layer manager has to be added to facilitate an intuitive way for manipulating multiple missions. At the same time, it must support the ability to tie the mission scripts to specific virtual AUVs via DIS application, site and entity ID fields. A possible representation of the mission layer manager is given in Figure 121. The "eye" icon allows the user to show or hide the mission on the display.

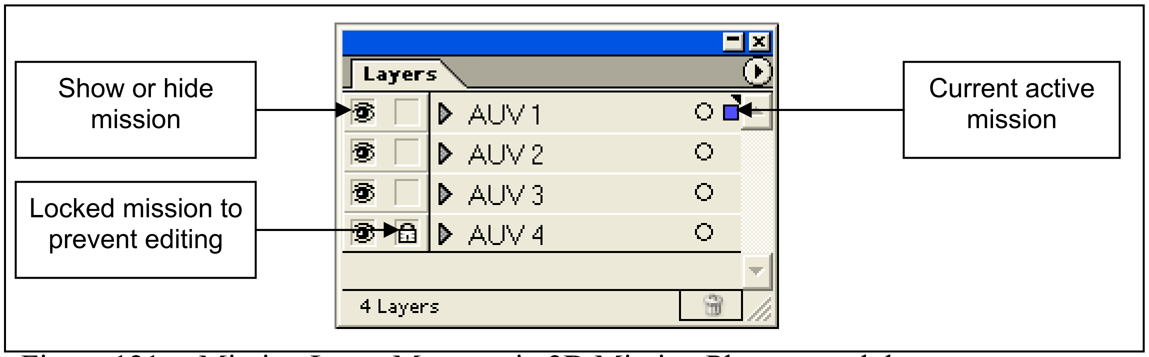


Figure 121. Mission Layer Manager in 2D Mission Planner module.

b. Animated Icons in 2D Mission Planner and Mission Command List

To promote a more intuitive user interface, animated icons can be introduced to depict the status of a command; e.g., when the propeller is turned on, it is animated. This gives a user a better appreciation of the current status of the mission commands. Similarly, the concept can be used in the 2D Mission Planner to depict important targets with a glowing red boundary. The concept of target decay (i.e., freshness of data) can be introduced using colors too; e.g., a new target has a solid color and as time passes, it becomes grayed out.

9. Distributed Robot (Execution) and Virtual Environment (Dynamics) Processes

At the moment, “Execution” and “Dynamics” processes are running on the same machine and Java Virtual Machine (JVM). The two processes use DIS multicast packets to talk to each other. In principle, both these processes are already network-capable. If both the processes are shifted to a server, it is possible to achieve a performance gain as it will offload the JVM. One approach is to implement the robot execution and virtual environment hydrodynamics using web services. The Workbench can toggle between running them on a server or local. Another approach is to use Jabber instant messaging. The AUV Workbench will package the active mission script using the method discussed in this thesis. The packaged mission script is then posted in a pre-defined chat-room. The “Execution” and “Dynamics” processes are Jabber-enabled so that they will pick up the packaged mission script and execute it.

With the server setup, the Execution and Dynamics processes are consolidated at a central location. This aids in development and testing.

The same server that functions as a Jabber server (for agent-to-agent communications through chat protocol) can also be configured as a web server (specifically Apache Tomcat).

10. Compression and Error-Correction Algorithms

Data compression is important in the operation of AUVs. Water density inhibits transmission of radio and light waves. Although sound travels quite well, currently achievable data transmission rates are poor in comparison to land-based communications.

A likely candidate for data compression is an in-house developed compression scheme, Cross Format Schema Protocol (XFSP) [Serin 2003] or XML Schema-based Binary Compression (XSBC). XSBC is schema-based XML binary serialization and compression.

In addition to compression, data error correction and recovery schemes can be introduced as acoustic shallow-water data transmissions are known to be unreliable and an autonomous entity will often experience problems when passing a message to its intended receiver. According to thesis work performed in 1995, Forward Error Correction (FEC) can reduce the number of required retransmissions by 3 to 15 percent. FEC is a "method of data encoding that gives the receiver the ability to correct data received in error up to a preset bound." FEC can be easily implemented, the most basic implementation requiring the use of a simple Hamming code. [Reimers 1995] As with the implementation of an XML-based mission control language, one goal of FEC is standardization of the underwater acoustic data communications community (after Reimers, 1995).

The study and introduction of encryption and decryption algorithms is important as AUVs are tasked to perform covert missions.

11. Mapping Capability in Mission Planner

The current version of the mission planner does not have any mapping capability. There are several commercial and Open Source products available to add mapping and geo-referencing capability to the Workbench. OpenMap is an Open Source Java Beans based toolkit for building applications and applets needing geographic information. Using OpenMap components, you can access data from legacy applications, in-place, in a distributed setting. At its core, OpenMap is a set of Swing components that understand geographic coordinates. The technology base underlying OpenMap was developed under government funding. From 1987 - 1992, BBN was involved in a DARPA collaborative mapping research project http://openmap.bbn.com/ (Accessed February 2004). Another open-source product is GeoTools. GeoTools is an open source Java toolkit for developing interactive geographical maps. The emphasis is on client side mapping applets that require little or no server side support. The main file format for the moment is the ESRI Shapefile (.shp). http://geotools.sourceforge.net/ (Accessed February 2004). There are two commercial solutions identified. They are iLog Jviews and ESRI Java MapObjects. ILOG JViews Maps Package provides a full range of features, including geo-referencing for easy placement of assets in proper locations, mix-and-match vector and raster data in the same map and the ability to handle multiple projections of the earth's surface It has built in load-on-demand for efficiently handling large sets of map data www.ilog.com (Accessed February 2004). An important feature as the Workbench acquires Geographic Information Systems (GIS) capability. ESRI Java MapObjects is a powerful collection of pure Java components that allows developers to build custom, cross platform, mapping and spatially enabled applications. With a robust collection of pure Java GIS and mapping components, including a suite of pre-defined visual JavaBeans, MapObjects—Java Edition provides developers with the tools to create client or server-side applications for stand-alone deployments or delivery over the Web www.esri.com (Accessed February 2004). A list of open-source and free GIS related software projects are available on http://opensourcegis.org/ (Accessed February 2004).

C. SUMMARY

Conclusion and future work recommendations are collected in this chapter. The goal of implementing a common platform for AUV mission planning and analysis has been achieved. At the same time, this thesis has shown that Jabber, an open-standards technology for instant messaging, is a viable solution to facilitate text and file messaging for humans as well as agent communications. Exemplars have demonstrated how in-mission and post-mission event monitoring by human operators can be achieved via simple web page, standard clients, or custom instant messaging client. Finally, the AUV Workbench is a potential tool for the development of multiple-AUV deployment concepts, tactics and doctrine.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. ACRONYMS AND ABBREVIATIONS

Acronym / Notation	Definition
2D	2 Dimensional
3D	3 Dimensional
API	Application Programming Interface
ARIES	Acoustic Radio Interactive Exploratory Server – NPS AUV
AUV	Autonomous Underwater Vehicle
DTD	Document Type Definition – XML
M&S	Modeling and Simulation
NPS	Naval Postgraduate School
REMUS	Remote Environmental Measurement UnitS
RF	Radio Frequency
URI	Uniform Resource Identifiers
URL	Uniform Resource Locator
URN	Uniform Resource Names
UUV	Unmanned Underwater Vehicle
X3D	Extensible 3D Graphics
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XMSF	Extensible Modeling and Simulation Framework
XSD	XML Schema Definition
XSL	Extensible Style Language
XSLT	Extensible Style Language Transformation
XTC	XML-based Tactical Chat

Table 37. Acronyms and abbreviations

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. LIST OF ARIES AUV-SPECIFIC EXECUTION-LEVEL COMMANDS

A. INTRODUCTION

This appendix consists of the list of ARIES AUV-specific execution level commands along with attribute types and example values. This language remains under developed; particularly with respect to in-water exception handling.

B. XML-BASED EXECUTION LEVEL COMMANDS

1. <Depth> Element

`<Depth>` sets commanded vehicle depth. Sample given below:

```
<Depth value="10"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	value	Depth value	Decimal	0.0	N

2. <EnterTube> Element

`<EnterTube>` commands the vehicle to enter a specified recovery tube. Vehicle initial position needs to be directly in front of opening, but heading can be off. Sample given below:

```
<EnterTube recoveryRange="5" recoveryHeading="270"/>
```

S/N	Command	Description	Type	Default	Required
1.	recoveryRange	Range from vehicle's current location to final recovery position.	Decimal	-	Y
2.	recoveryHeading	Recovery heading (must match orientation of the tube).		-	Y
3.	timeout	Specifies a max allowable time for the command (negative means no limit) before failure.	Decimal	-1.0	N

3. <FollowLight> Element

`<FollowLight>` commands the vehicle to follow a light source to a recovery location. Sample given below:

```
<FollowLight/>
```

S/N	Command	Description	Type	Default	Required
1.	timeout	Specifies a max allowable time for the command (negative means no limit) before failure.	Decimal	-1.0	N

4. <GpsFix> Element

`<GpsFix>` orders the vehicle to surface for a GPS fix or resume mission after a obtaining a GPS fix. Sample given below:

```
<GpsFix status=”complete” timeOut=”5.0”/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	status	Operational status of GPS unit.	GpsFixStatus = {“start”, “inProgress”, “complete”, “failed”} Enumerated list of possible GPS fix statuses.	start	N
2.	timeout	Specifies a max allowable time for the command (negative means no limit) before failure.	Decimal	-1.0	N

5. <Heading> Element

`<Heading>` sets commanded vehicle heading (disables waypoint or recovery control). Sample given below:

```
<Heading value="270"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	value	Course heading.	Decimal	0.0	N

6. <Help> Element

`<Help>` causes a list of valid commands to be printed to the console (if available). Sample given below:

```
<Help/>
```

7. <Hover> Element

`<Hover>` commands the vehicle maintain position at a specified (x,y) position. It can include heading and depth command. Sample given below:

```
<Hover x="200" y="100" z="15" heading="270" standoff="2.5"/>
```

S/N	Attribute	Description	Format	Default	Required
1.	x	x-coordinate.	decimal	-	N
2.	y	y-coordinate.	decimal	-	N
3.	z	z-coordinate.	decimal	-	N
4.	heading	Course heading.	decimal	-	N
5.	standoff	Stand-off distance.	decimal	-	N
6.	altitudeControl	Determines whether z is depth or height above the bottom.	boolean	false	N
7.	timeout	timeOut attribute specifies a max allowable time for the command (negative means no limit) before failure.	decimal	-1.0	N

8. <Lateral> Element

`<Lateral>` sets both lateral thrusters to cause vehicle to slide right or left (turns off all automatic control modes). Sample given below:

```
<Lateral speed="10"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	speed	Speed of thrusters.	Decimal	0.0	N

9. <MissionScript> Element

`<MissionScript>` loads a new mission script from a specified file. Sample given below:

```
<MissionScript fileName="sample.xml"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	path	Directory or full path information.	String		N
2.	filename	File name.	String		Y

10. <Pause> Element

`<Pause>` temporarily suspends vehicle operation (for bench test or virtual world use only), useful for getting evaluation checkpoints during testing. Sample given below:

```
<Pause/>
```

11. <Planes > Element

`<Planes>` sets bow and/or stern plane deflection angle (turns off all automatic control modes). Sample given below:

```
<Planes which="stern" value="-25"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	which	Set planes	AvailablePlanes = {“bow”, “stern”, “both”} Enumerated list of manually settable control planes.	both	N
2.	value	Angle of deflection.	Decimal	0.0	N

12. <Position> Element

`<Position>` updates of vehicle position in the world (new navigation fix has been obtained). It sets GPS zero point if not previously done. Sample given below:

```
<Position x="12" y="55" depth="5"/>
```

S/N	Attribute	Description	Format	Default	Required
1.	x	x-coordinate.	decimal	-	Y
2.	y	y-coordinate.	decimal	-	Y
3.	depth	Depth data.	decimal	-	N

13. <Propeller> Element

`<Propeller>` manually set one or both propeller speeds. Sample given below:

```
<Propeller which="port" rpm="10"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	which	Location.	AvailablePropellers = {"port", "starboard", "center", "both"} Enumerated list of manually settable propellers.	both	N
2.	Rpm	Speed.	Decimal	-	Y

14. <Quit> Element

`<Quit>` ends the vehicle mission after zeroing all control settings (does not initiate surfacing procedure). Sample given below:

```
<Quit/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	mode		ExitModes = {"normal", "missionAbort", "systemAbort", "recallAbort" } Enumerated list of possible mission ending modes.	-	N

15. <RealTime> Element

`<RealTime>` causes execution to run in realtime (or turns realtime execution off). Sample given below:

```
<RealTime/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	set	Turn on or off.	Boolean	true	N

16. <ResetTime> Element

`<ResetTime>` resets the vehicle clock time to a specified value. Sample given below:

```
<ResetTime value="1.0"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	Value		Decimal	0.0	N

17. <Rotate> Element

`<Rotate>` sets both lateral thrusters to cause vehicle to rotate (turns off all automatic control modes). Sample given below:

```
<Rotate speed="-10"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	speed	Speed of thrusters.	Decimal	0.0	N

18. <Rudder> Element

`<Rudder>` sets rudder deflection (turns off all automatic control modes). Sample given below:

```
<Rudder value=”0.5”/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	value	Angle of deflection.	Decimal	0.0	N

19. <Sonar> Element

`<Sonar>` commands the vehicle to assume a specified fixed station relative to a sonar target. Sample given below:

```
<Sonar sonarHardware="ST1000" scanMode="manual" bearing="180"
bearingType="relative"/>
```

S/N	Command	Description	Type	Default	Required
1.	sonarHardware	Sonar models that may be installed.	SonarHardwareModels = {“ST1000”, “ST725”};	st725	N
2.	Mode	Sonar scan modes.	SonarScanModes= {“scan”, “track”, “trackWhileScan”, “manual”}	scan	N
3.	bearing	Direction.	decimal	0.0	N
4.	bearingType	Angles measured from bow/north clockwise when viewed from above. Matches a standard compass rose.	BearingTypes= {“relative”, “true”, “magnetic”}	relative	N

20. <Standoff> Element

`<Standoff>` resets the acceptable standoff radius in meters around hover-points and waypoints. Sample given below:

```
<Standoff range=”15.0”/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	Range	Stand-off distance.	Decimal	2.5	N

21. <TakeStation> Element

`<TakeStation>` commands the vehicle to assume a specified fixed station relative to a sonar target. Sample given below:

```
<TakeStation sonarScanMode="target" targetRange="5"
targetBearing="90" commandRange="5" commandBearing="45"
heading="270"/>
```

S/N	Command	Description	Type	Default	Required
1.	sonarScanMode	Determines whether the vehicle will maintain station by sonar scanning the entire target or just the edge.	TargetTrackModes = {"targetEdge", "target"} Enumerated list of sonar target tracking modes.	targetEdge	N
2.	targetRange	Approximate range to target to enable sonar to initially acquire (not required if vehicle is already tracking).	Decimal	-	N
3.	targetBearing	Approximate bearing of target to enable sonar to initially acquire (not required if vehicle is already tracking).	Decimal	-	N
4.	commandRange	Commanded range to for vehicle to remain from the target.	Decimal	-	Y
5.	commandBearing	Commanded bearing to the target for the vehicle to maintain.	Decimal	-	Y
6.	heading	Course heading.	Decimal	-	N
7.	timeout	Specifies a max allowable time for the command (negative means no limit) before failure.	Decimal	-1.0	N

22. <Thrusters> Element

`<Thrusters>` enables or disables the vehicle's vertical and lateral thrusters (can be overridden by some control commands). Sample given below:

```
<Thrusters on="true"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	On	Specifies a max allowable time for the command (negative means no limit) before failure.	Boolean	true	N
2.	which	Location.	AvailableThrusters = {"lateral", "vertical", "bowLateral", "sternLateral", "bowVertical", "sternVertical"} Enumerated list of manually settable thrusters.	-	N

23. <TimeStep> Element

`<TimeStep>` resets the elapsed time for each closed loop control cycle (default is 0.1sec or 10 hz). Faster on-board computers and faster analog-to-digital (A/D and D/A) conversions permits shorter timestep periods. Sample given below:

```
<TimeStep period="0.5"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	Period	Loop interval.	Decimal	0.1	N

24. <Trace> Element

`<Trace>` turns vehicle trace feature on or off. Sample given below:

```
<Trace/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	set	Turn on or off.	Boolean	true	N

25. <Wait> Element

`<Wait>` causes the vehicle to wait a specified time before beginning execution of the next command. Sample given below:

```
<Wait time="10"/>
```

S/N	Command	Description	Format/Type	Default	Required
1.	absolute	Relative or absolute	Boolean	false	N
2.	time	Time to wait.	Decimal	-	Y

26. <Waypoint> Element

`<Waypoint>` commands the vehicle to transit to a specified location. Vehicle will not stop when location reached. Sample given below:

```
<Waypoint x="25" y="50" z="75" obtainGpsFix="false"/>
```

S/N	Attribute	Description	Format	Default	Required
1.	x	x-coordinate.	decimal	-	Y
2.	y	y-coordinate.	decimal	-	Y
3.	z	z-coordinate.	decimal	-	Y
4.	rpm	Speed	decimal	-	N
5.	altitudeControl	Determines whether z is depth or height above the bottom.	boolean	false	N
6.	timeOut	timeOut attribute specifies a max allowable time for the command (negative means no limit) before failure.	decimal	-1.0	N
7.	obtainGpsFix	Cause the vehicle to surface to obtain a GPS fix enroute to the next waypoint.	boolean	false	N
8.	fixDuration	determines how long the vehicle will remain surfaced to obtain a gps fix if the obtainGpsFix attribute is true.	decimal	-	N

APPENDIX C. CDROM MATERIAL

A. DIRECTORY AND FILE STRUCTURE

1. Documentation

Directory location: `<CDRom>\documentation`

S/N	Directory	Filename	Description
1.	\	04Mar_Lee_AUVWorkbench.doc	Thesis (in Microsoft WinWord format).
2.	\	04Mar_Lee_AUVWorkbench.pdf	Thesis (in Adobe Acrobat format).
3.	\	AUVWorkbench.ppt	AUV Workbench presentation (in Microsoft Powerpoint format). AUV Workbench icons and component chart (in Microsoft Powerpoint format).
4.	\	AgentJabber.ppt	Presentation slides using Agent Seminar on 17 Feb 2004 (in Microsoft Powerpoint format).
5.	\	XTC.ppt	XML-based Tactical Chat presentation (in Microsoft Powerpoint format).
6.	\reference	*	Reference material used in the conduct of this thesis.

2. AUV Workbench Application

Directory location: `<CDRom>\auv\Workbench\`

S/N	Directory	Sub-directory and files	Description
1.	\src	im	Java source code to the Jabber Instant Messaging and XTC Event Monitor modules.
2.		main	Java source code to the main user interface and 3D Visualization module.
3.		mission	Java source code to the two-dimensional mission planner module.
4.		util	Java source code for Common utilities.
5.		web	Java source code to the web server
6.	\bin	im	Java classes to the Jabber Instant Messaging and XTC Event Monitor modules.
7.		main	Java classes to the main user interface and 3D Visualization module.
8.		mission	Java classes to the two-dimensional mission planner module.

S/N	Directory	Sub-directory and files	Description
9.		util	Java classes for Common utilities.
10.		web	Java classes to the web server
11.		image	Icon and splash-screen image files (in GIF, JPEG and PNG formats).
12.		sound	Sound files (in .WAV).
13.		META-INF\	Contains `manifest.mf` for the JAR.
14.	\doc	*	HTML documentation.
15.	\dynamics	*	Java application to the hydrodynamics for the virtual environment.
16.	\execution	*	C++ application to the AUV robot execution.
17.	\Java Execution	*	Java application to the AUV robot execution.
18.	\javadocs	., \im, \main, \mission, \util, \web, \xsbc	Java documentation of the source code.
19.	\lib	*	Java libraries.
20.	\Models	*	Sample VRML examples.
21.	\Scripts	*	Mission scripts.

B. MAIN APPLICATION

The `main` package is the main user interface for the rendering of the entire user interface including the placements of the user interfaces for the various modules and the 3D visualization.

Directory location: `<CDRom>\auv\Workbench\src\main`

S/N	Filename	Description
1.	AMVW.java	Main user interface for AUV Workbench.
2.	AUV.java	Data structure for AUV information (not used). This is to be used for multiple AUVs in the same scene.
3.	AUVWorkbenchConfig.java	AUV Workbench configuration data structure.
4.	ConfigApp.java	Application configuration data structure. Used by configurable toolbar
5.	Const.java	Application global constants.
6.	DynamicsExecutionThread.java	Invoke a separate process dynamics (located in `..\dynamics\dynamics`).
7.	UITable.java	User interface to display data in a tabular format.

S/N	Filename	Description
8.	VrmlLoader.java	Xj3D loader for VRML models.
9.	X3DLoader.java	Xj3D loader for X3D models (not used).

C. MISSION PLANNING

The `mission` package is responsible to render two-dimensional mission planner view on the top-right display pane.

Directory location: `<CDRom>\auv\Workbench\src\mission`

S/N	Filename	Description
1.	Mission.java	Data structure to store Mission information.
2.	MissionBoundBoxView.java	User interface to define the mission bounding box (area of interest) (Not used).
3.	MissionCommand.java	Generic mission command data structure.
4.	MissionDepth.java	Mission Depth command data structure. It defines commanded vehicle depth
5.	MissionDialog.java	Mission information dialog user interface.
6.	MissionDrawArea.java	Drawing canvas/area to display Mission Script graphically.
7.	MissionEnterTube.java	Mission EnterTube command data structure. It commands the vehicle to enter a specified recovery tube. Vehicle should be directly in front of opening, but heading can be off.
8.	MissionFollowLight.java	Mission FollowLight command data structure. It commands the vehicle to follow a light source to a recovery location.
9.	MissionHeading.java	Mission Heading command data structure. It sets commanded vehicle heading (disables waypoint or recovery control).
10.	MissionHelp.java	Mission Help command data structure. It causes a list of valid commands to be printed to the console (if available).
11.	MissionHover.java	Mission Hover command data structure. It commands the vehicle maintain position at a specified (x,y) position. It can include heading and depth command.
12.	MissionInputOneView.java	User interface to capture a single value input (boolean, integer) from user. It is invoked by MissionDialog.
13.	MissionLateral.java	Mission Lateral command data structure. It sets both lateral thrusters to cause vehicle to slide right or left (turns off all automatic control modes).

S/N	Filename	Description
14.	MissionListCellRenderer.java	Customized cell rendering in a JList (e.g., loading of icons and setting of colors).
15.	MissionListView.java	User interface to display Mission commands in a Listbox. Sends ACTION_PERFORMED event for double-click and ENTER key.
16.	MissionMissionScript.java	Mission Script command data structure. Loads a new mission script from a specified file.
17.	MissionPause.java	Mission Pause command data structure. It temporarily suspends vehicle operation (for bench test or virtual world use only); useful for getting evaluation checkpoints during testing.
18.	MissionPlanes.java	Mission Planes command data structure. Set bow and/or stern plane deflection angle (turns off all automatic control modes).
19.	MissionPoint.java	Mission Point command data structure. MissionHover, MissionPosition, MissionWaypoint inherit from this.
20.	MissionPointView.java	Mission Point user interface to manipulate MissionPoint data (includes Hover, Position, Waypoint).
21.	MissionPosition.java	Mission Position command data structure. It updates of vehicle position in the world (new navigation fix has been obtained). It sets GPS zero point if not previously done.
22.	MissionPropeller.java	Mission Propeller command data structure. It manually set one or both propeller speeds.
23.	MissionQuit.java	Mission Quit command data structure. It ends the vehicle mission after zeroing all control settings (does not initiate surfacing procedure).
24.	MissionRealtime.java	Mission RealTime Command Information. It causes execution to run in realtime (or turns realtime execution off).
25.	MissionResetTime.java	Mission ResetTime command data structure. It resets the vehicle time to a specified value.
26.	MissionRotate.java	Mission Rotate command data structure. It sets both lateral thrusters to cause vehicle to rotate (turns off all automatic control modes).
27.	MissionRudder.java	Mission Rudder command data structure. It sets rudder deflection (turns off all automatic control modes).
28.	MissionSonar.java	Mission Sonar command data structure. It commands the vehicle to assume a specified fixed station relative to a sonar target.
29.	MissionSpeed.java	Mission Speed command data structure (not used).

S/N	Filename	Description
30.	MissionStandoff.java	Mission Standoff command data structure. It resets the acceptable standoff radius in meters around hover-points and waypoints.
31.	MissionTakeStation.java	Mission TakeStation command data structure. It commands the vehicle to assume a specified fixed station relative to a sonar target.
32.	MissionThruster.java	Mission Thruster command data structure. It enables or disables the vehicle's vertical and lateral thrusters (can be overridden by some control commands).
33.	MissionTimeStep.java	Mission Timesetp command data structure. It resets the elapsed time for each closed loop control cycle (default is 0.1sec or 10 Hz).
34.	MissionTrace.java	It turns vehicle trace feature on or off.
35.	MissionViewer.java	2D mission script viewer/planner.
36.	MissionViewerConfig.java	2D mission script viewer/planner configuration file (Not implemented yet).
37.	MissionWait.java	Mission Wait command data structure. It causes the vehicle to wait a specified time before beginning execution of the next command.
38.	MissionWaypoint.java	Mission Waypoint command data structure. It commands the vehicle to transit to a specified location.
39.	TargetMine.java	Target mine data structure.

D. JABBER INSTANT MESSAGING

The `im` package is used for standard Jabber instant messaging. It also implements the XTC Event Monitoring module (including the triggering of watch events and raising of alerts).

Directory location: `<CDRom>\auv\Workbench\src\im`

S/N	Filename	Description
1.	AgentConfig.java	Data structure to store agent configuration that has been loaded from an XML file.
2.	Alert.java	Data structure to store alert. AlertSound, AlertURL and AlertVisual inherit from this class.
3.	AlertSound.java	Data structure for Sound Alerts, e.g., `<Alert type="sound" src="sound/beep.au"/>`.
4.	AlertURL.java	Data structure for Hyperlink Alerts, e.g., `<Alert type="url" src="http://www.google.com"/>`

5.	AlertVisual.java	Data structure for Visual Alerts, e.g., `<Alert type="visual" src="image/mine.gif"/>`
6.	IMConfig.java	Instant messaging session object.
7.	Monitor.java	Event monitoring criteria for Jabber messages.
8.	UIAgent.java	Event Monitoring/Jabber Instant Messaging User Interface.
9.	WatchEvent.java	Data structure to store Watch Event and its corresponding alerts/actions.

E. WEB

The web server module is implemented in the `web` package.

Directory location: `<CDRom>\auv\Workbench\src\web`

S/N	Filename	Description
1.	HandleRequest.java	Thread to handle incoming web server requests.
2.	HTTPServer.java	Web server to receive HTTP requests.
3.	PostForm.java	HTTP POST data structure.
4.	RequestHTTP.java	Processing of incoming web server requests.

F. UTILITIES

Common utilities and procedures are kept in `util` package.

Directory location: `<CDRom>\auv\Workbench\src\util`

S/N	Filename	Description
1.	AgentPayload.java	Data structure to store binary file data in XHTML portion of Jabber message.
2.	FileFilterEx.java	Define a file filter (extension, description) in drop-down combo-box
3.	FontDialog.java	Selection of font type or allow typed-in text string, e.g., used in drawing application.
4.	IconFileView.java	Display an icon for a particular file types.
5.	ImageDisplay.java	Image viewer for the following formats: BMP, GIF, PNG, JPEG and SVG (using Batik).
6.	NumericInputHandler.java	To restrict the no. of characters permitted in the JTextField.
7.	SortedList.java	Sorted JList component.
8.	SplashScreen.java	Splash screen.
9.	SystemMedia.java	System Utilities to manipulate media files e.g.,

S/N	Filename	Description
		sound.
10.	SystemUtil.java	System Utilities to perform file copy, extract file name, directory, name only, property management and screen-capture.
11.	SystemUtilX.java	Extra System Utilities to perform base-64 encoding & decoding, ZIP, GZIP and HTTP file retrieval

G. LIBRARIES

List of required libraries provided from external sources.

Directory location: `<CDRom>\auv\Workbench\lib`

S/N	Library	Version	Filename	Description
1.	Apache Ant	1.6.0	ant.jar, optional.jar, xercesImpl.jar, xml-apis.jar	Java-based build tool.
2.	Apache SOAP	2.3.1	soap.jar	Base-64 encoding and decoding.
3.	Apache Xerces	2.5.0	xmlParserAPIs.jar, xml-apis.jar, xercesImpl.jar	XML parsing.
4.	Apache Xalan	2.5.0	xalan.jar	XML transformation,
5.	Batik	1.5.0	batik-awt-util.jar, batik-bridge.jar, batik-css.jar, batik-dom.jar, batik-ext.jar, batik-gvt.jar, batik-parser.jar, batik-script.jar, batik-svg-dom.jar, batik-svggen.jar, batik-swing.jar, batik-util.jar, batik-xml.jar, js.jar	A Java-based toolkit for apps that want to use images in the SVG format for viewing, creation and manipulation.
6.	Extensible Java 3D	M8	aviatrix3d-all.jar, gnu-regexp-1.0.8.jar, httpclient.jar, j3d-org-images.jar, j3d-org.jar, Jama.jar, js.jar, JXInput.jar, uri.jar, vlc_uri.jar, vrml97.jar, xj3d-all.jar	Display of 3D VRML and X3D models
7.	Jivesoftware SMACK APIs	1.2.1	smack.jar, smackx.jar.	XMPP communications.
8.	dis-java-vrml	-	dis-java-vrml.jar	Distributed Interactive Simulation.

H. CONFIGURATION FILE

The AUV Workbench configuration file, `AUVWorkbenchConfiguration.xml,` is located in directory `<CDRom>\auv\Workbench\bin`.

APPENDIX D. AUV WORKBENCH DEVELOPER AND USER GUIDE

A. SETUP

This document explains how to install the current version (as of March, 2004) of the AUV Workbench application. This setup procedure assumes the user is running in a Windows environment without any of the needed components installed.

1. Download and install Sun Java SDK 1.4.2 (Available at http://java.sun.com/j2se/1.4.2/download.html. Accessed on 15 February 2004). Ensure that "JAVA_HOME" is set.

2. Download and install Sun J3D API 1.3.1 (Available at http://java.sun.com/products/java-media/3D/download.html. Accessed on 15 February 2004).

3. Download and install ANT 1.6.0 or above. This is required to build the AUV Workbench application. (Available at http://ant.apache.org. Accessed on 15 February 2004).

4. Download and install Distributed Interactive Simulation module. Ensure that "dis-java-vrml.jar" (only file required) is installed in "C:\vrtp" and it is set in the "CLASSPATH".

5. Download and unzip AUV Workbench application (source and executable) into directory "C:\auv\Workbench".

6. Download and install list of applications and tools in Section C.

B. HOW TO RUN IT

By default, the AUV Workbench application shall be located in `C:\auv\Workbench`. To run it, go to `C:\auv\Workbench\bin` directory and double-click on `run.bat`.

C. HOW TO COMPILE IT

To compile and build the AUV Workbench application, go to `C:\auv\Workbench\` directory and double-click on `antbuild.bat`.

D. TOOLS AND APPLICATIONS

List of tools and useful applications:

S/N	Name	Version	Description	Available at
1.	jEdit	4.1	Java text editor.	http://www.jedit.org
2.	Mozilla	1.6	Internet browser.	http://www.mozilla.org
3.	ParallelGraphics Cortona	4.2	ParallelGraphics browser VRML plugin.	http://www.parallelgraphics.com/products/cortona
4.	Rhymbox	1.6	Jabber instant messaging client.	http://www.rhymbox.com
	ParallelGraphics VrmlPad	2.0	Vrml editor.	http://www.parallelgraphics.com/products/vrmlpad
5.	X3D Edit	2.4	X3D Graphics editor.	http://www.web3d.org
6.	Xj3D	M8	Java-based VRML and X3D loader.	http://www.xj3d.org

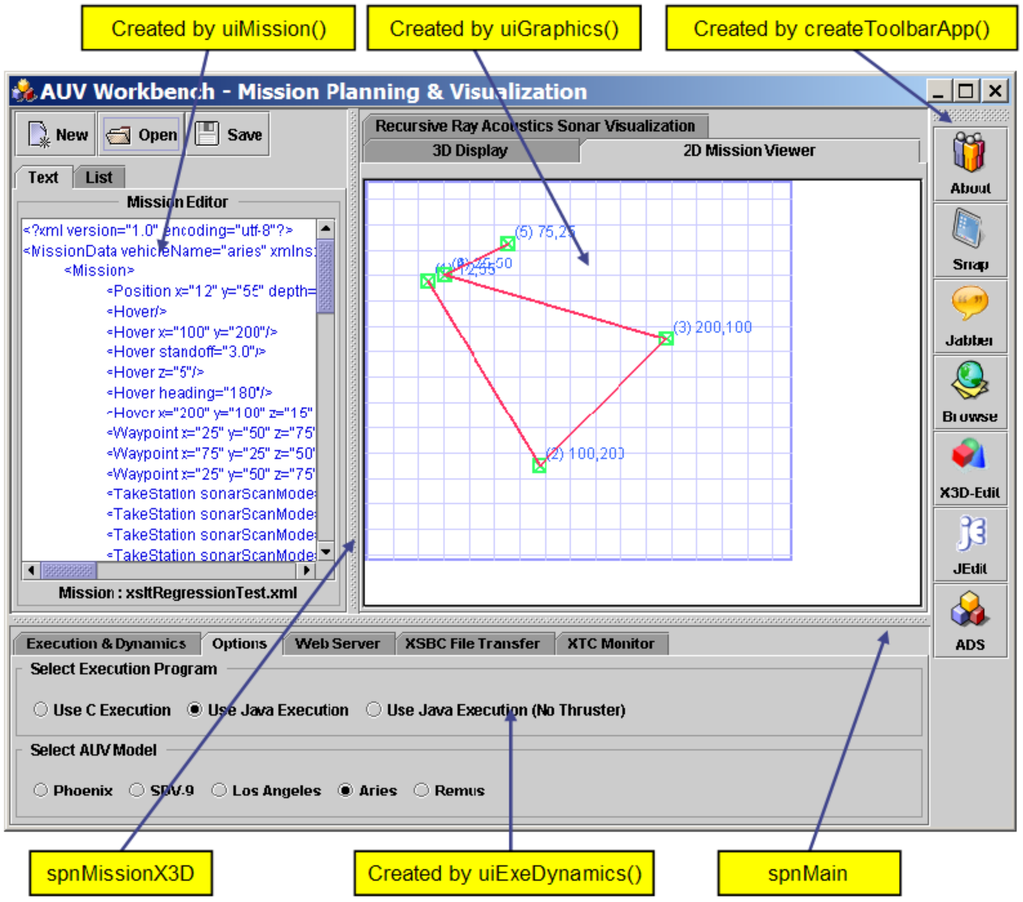
E. FREQUENTLY ASKED QUESTIONS (FAQ)

1. Unable to Start AUV Application

- Install Java JDK 1.4.2 .
- Install Java 3D .
- Go to AUV's "\bin" directory, double-click on "run.bat".
- List of files that are required by Java3D:
- J3D.dll, J3DUtils.dll, j3daudio.dll located in %JAVA_HOME/jre/bin.

- vecmath.jar, j3dcore.jar, j3dutils.jar, j3daudio.jar located in %JAVA_HOME%/jre/lib/ext.

F. COMPONENT CHART



```
<?xml version="1.0" encoding="utf-8"?>
<MissionData vehicleName="aries" xmlns:
        <Mission>
                <Position x="12" y="55" depth=
                <Hover/>
                <Hover x="100" y="200"/>
                <Hover standoff="3.0"/>
                <Hover z="5"/>
                <Hover heading="180"/>
                <Hover x="200" y="100" z="15"
                <Waypoint x="25" y="50" z="75"
                <Waypoint x="75" y="25" z="50"
                <Waypoint x="25" y="50" z="75"
                <TakeStation sonarScanMode
                <TakeStation sonarScanMode
                <TakeStation sonarScanMode
                <TakeStation sonarScanMode
```

Mission : xsltRegressionTest.xml

Select Execution Program

Use C Execution [✓ Use Java Execution] Use Java Execution (No Thruster)

Select AUV Model

Phoenix SDV-9 Los Angeles [✓ Aries] Remus

Figure 122. Main Application User Interface.

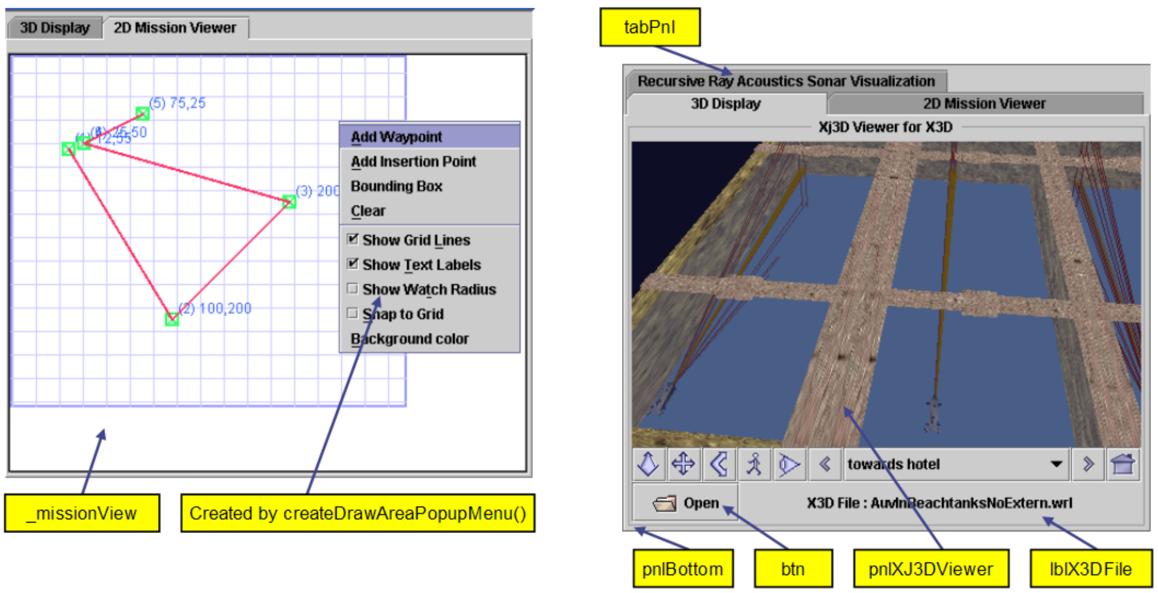


Figure 123. 2D Mission Planning and 3D Visualization User Interface.

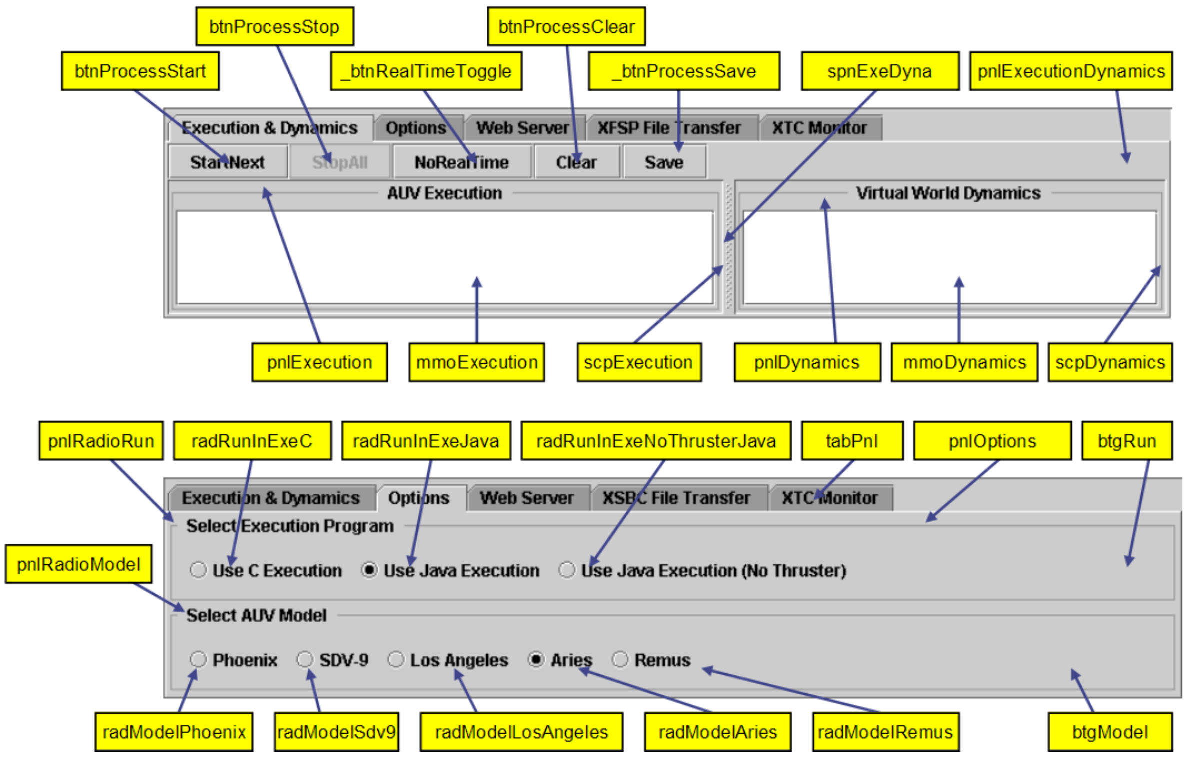


Figure 124. Execution and Hydro-Dynamics User Interface.

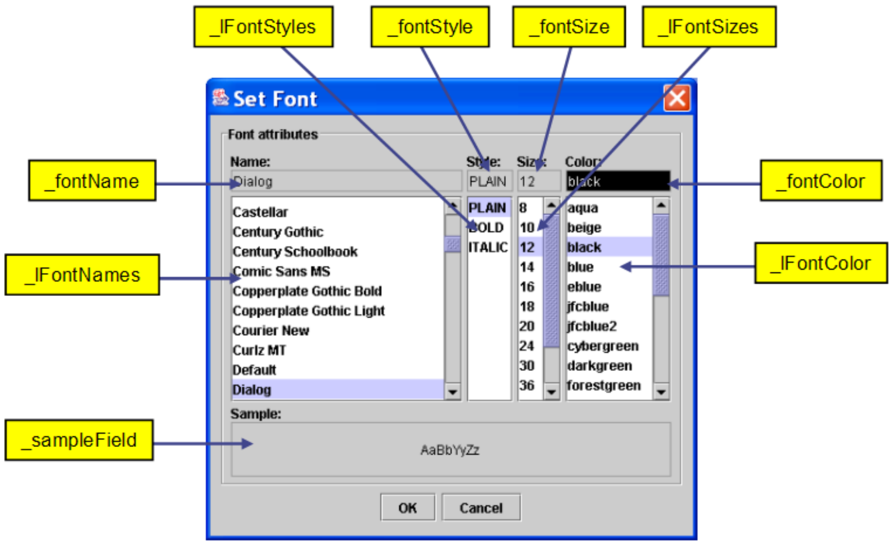


Figure 125. Font Dialog User Interface.

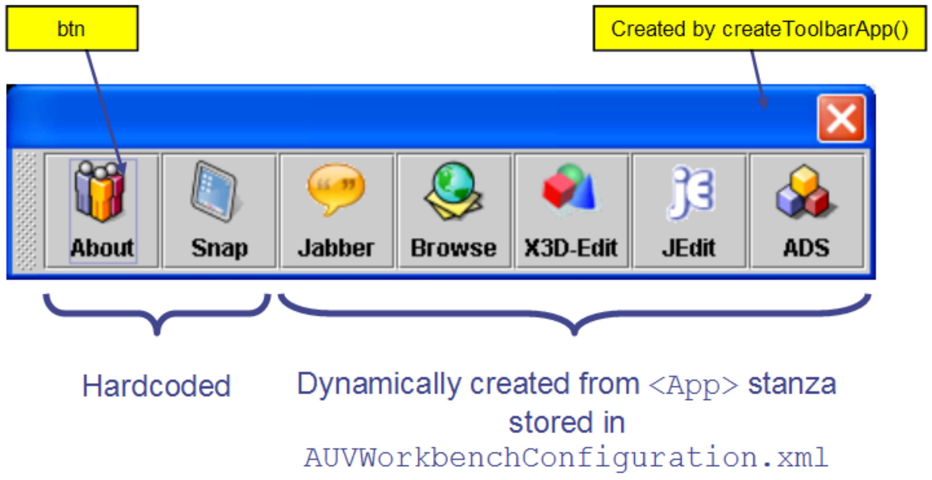
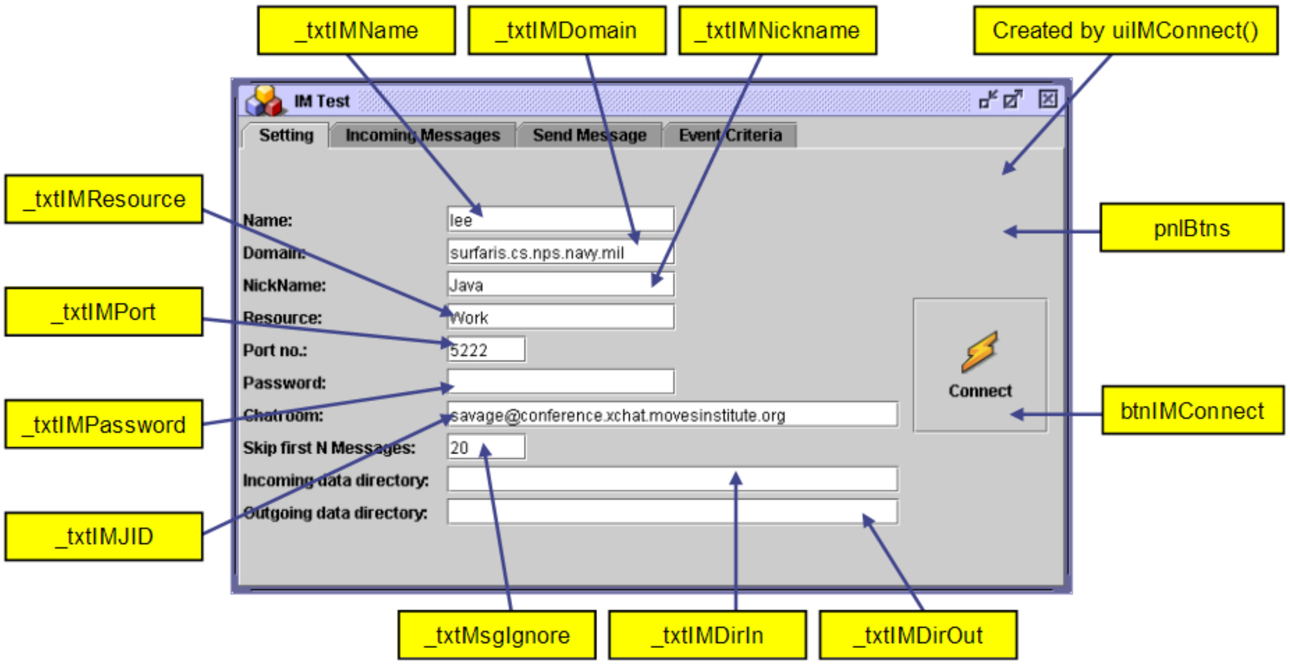
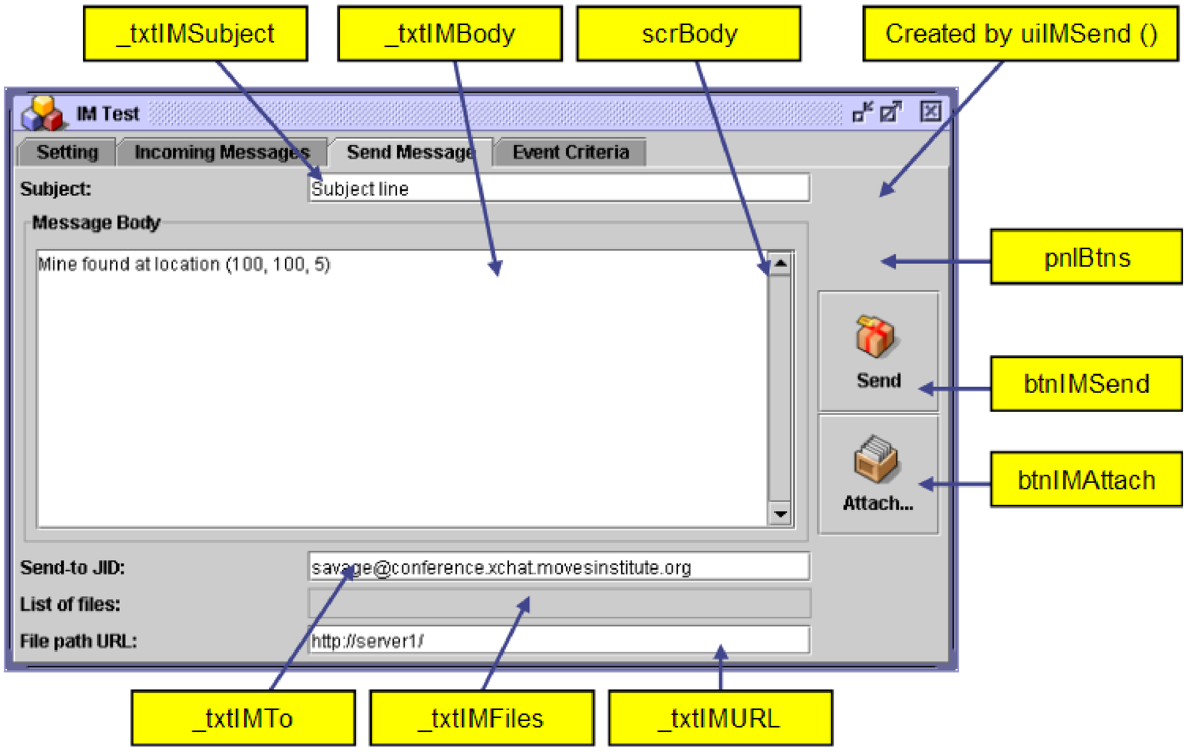

btn
Created by createToolbarApp()
About
Snap
Jabber
Browse
X3D-Edit
JEdit
ADS
Hardcoded
Dynamically created from <App> stanza stored in AUVWorkbenchConfiguration.xml

Figure 126. Application Toolbar.



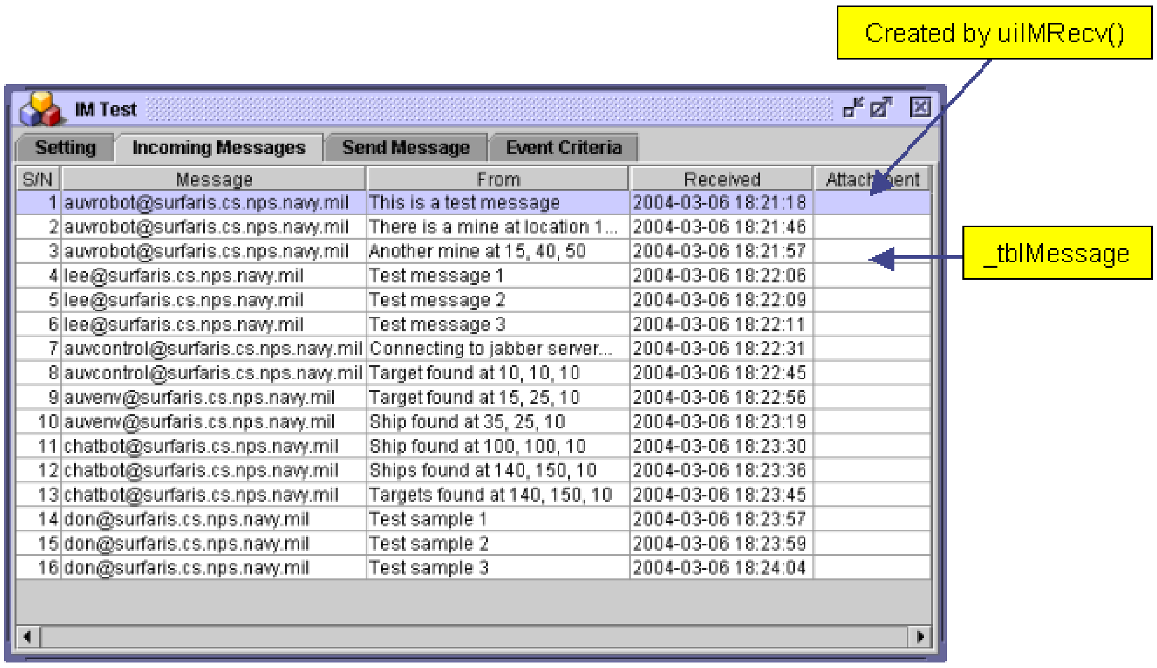
Name: lee
Domain: surfaris.cs.nps.navy.mil
NickName: Java
Resource: Work
Port no.: 5222
Password:
Chatroom: savage@conference.xchat.movesinstitute.org
Skip first N Messages: 20
Incoming data directory:
Outgoing data directory:

Figure 127. Customized Jabber Client – Message Settings Module.



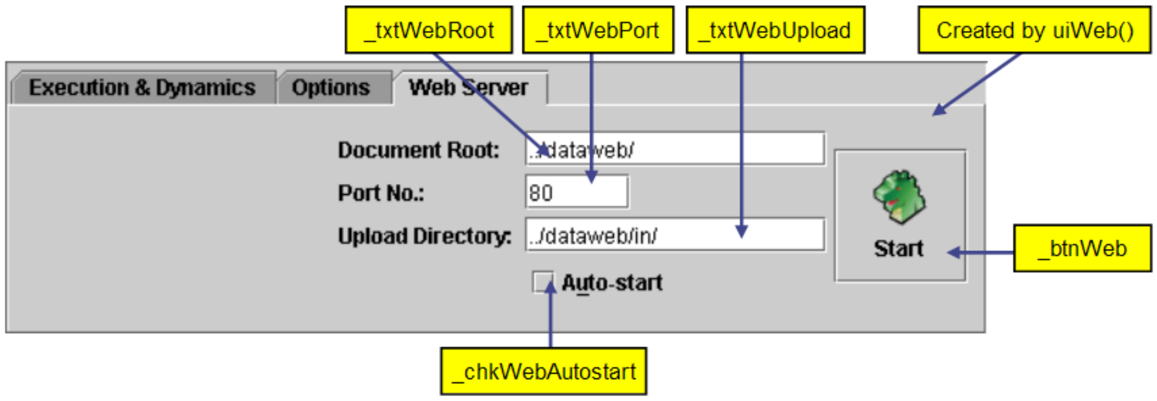
Subject: Subject line
Message Body
Mine found at location (100, 100, 5)
Send-to JID: savage@conference.xchat.movesinstitute.org
List of files:
File path URL: http://server1/

Figure 128. Customized Jabber Client – Message Send Module.



S/N	Message	From	Received	Attachment
1	auvrobot@surfaris.cs.nps.navy.mil	This is a test message	2004-03-06 18:21:18	
2	auvrobot@surfaris.cs.nps.navy.mil	There is a mine at location 1...	2004-03-06 18:21:46	
3	auvrobot@surfaris.cs.nps.navy.mil	Another mine at 15, 40, 50	2004-03-06 18:21:57	
4	lee@surfaris.cs.nps.navy.mil	Test message 1	2004-03-06 18:22:06	
5	lee@surfaris.cs.nps.navy.mil	Test message 2	2004-03-06 18:22:09	
6	lee@surfaris.cs.nps.navy.mil	Test message 3	2004-03-06 18:22:11	
7	auvcontrol@surfaris.cs.nps.navy.mil	Connecting to jabber server...	2004-03-06 18:22:31	
8	auvcontrol@surfaris.cs.nps.navy.mil	Target found at 10, 10, 10	2004-03-06 18:22:45	
9	auvenv@surfaris.cs.nps.navy.mil	Target found at 15, 25, 10	2004-03-06 18:22:56	
10	auvenv@surfaris.cs.nps.navy.mil	Ship found at 35, 25, 10	2004-03-06 18:23:19	
11	chatbot@surfaris.cs.nps.navy.mil	Ship found at 100, 100, 10	2004-03-06 18:23:30	
12	chatbot@surfaris.cs.nps.navy.mil	Ships found at 140, 150, 10	2004-03-06 18:23:36	
13	chatbot@surfaris.cs.nps.navy.mil	Targets found at 140, 150, 10	2004-03-06 18:23:45	
14	don@surfaris.cs.nps.navy.mil	Test sample 1	2004-03-06 18:23:57	
15	don@surfaris.cs.nps.navy.mil	Test sample 2	2004-03-06 18:23:59	
16	don@surfaris.cs.nps.navy.mil	Test sample 3	2004-03-06 18:24:04	

Figure 129. Customized Jabber Client – Message Send Receive.



Document Root: ../dataweb/
Port No.: 80
Upload Directory: ../dataweb/in/
☐ Auto-start

Figure 130. Web Server.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. PROCEDURE TO PACKAGE BINARY DATA

```
//---------------------------------------------------------------------------
/**
 * base64 encode data from a file and create XML Document
 * perform zipping based on file formats (see SystemUtil.fileCanZip)
 *
 * @param sTagName tag name to be used
 * @param lstAP list of AgentPayload objects
 *        (file name, description & URLs)
 * @param flagZip whether to gzip zippable files
 * @return created XML document
 */
public static org.w3c.dom.Document encodeDataToXML(String sTagName,
                                                   ArrayList lstAP,
                                                   boolean flagZip) {
  org.w3c.dom.Document xmlDoc  = null;
  org.w3c.dom.Element eWrapper = null;
  org.w3c.dom.Element ePayload = null;
  org.w3c.dom.Element eComment = null;
  org.w3c.dom.Element eURL     = null;
  boolean fZipit               = false; // false; //
  boolean fCdata               = false; // false; //
  int numPayload               = 0;

  AgentPayload objAP           = null;
  String srcFile               = "";
  String sDesc                 = "";
  String arrURL[];

  if (sTagName.length()>0) { // root to append <AgentPayload> elements
    //Create an XML Document
    try {
      DocumentBuilderFactory dbFactory =
                                   DocumentBuilderFactory.newInstance();
      DocumentBuilder docBuilder       = dbFactory.newDocumentBuilder();
      xmlDoc                           = docBuilder.newDocument();
    } catch(Exception ex) {
      System.out.println("encodeDataToXML() Error " + ex.getMessage() );
    }
    // <AgentJabber> wrapper around 1 or many <AgentPayload> tags
    eWrapper = xmlDoc.createElement(sTagName);

    // loop through list of AgentPayload objects
    for (int iAP=0; iAP<lstAP.size(); iAP++) {
      objAP   = (AgentPayload) lstAP.get(iAP);
      srcFile = objAP.getFilepath();
      sDesc   = objAP.getDesc();
      arrURL  = objAP.getURLs();

      if ( srcFile.length()>0 )  {
        // check whether file is zippable
        fZipit = SystemUtil.fileCanZip(srcFile);
        try {
          byte[] originalBytes = null;
          byte[] zippedBytes   = null;

          originalBytes = SystemUtil.fileRead(srcFile);
          if (originalBytes.length<=DEFAULT_PAYLOAD_SIZE) {
            fCdata = true;
          }
          //-------------------------------------------------------------
          // perform Gzip if file format is zippable & flagZip is SET
          if (fZipit && flagZip) {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            GZIPOutputStream zos       = new GZIPOutputStream(baos);
            zos.write(originalBytes);
            zos.flush();
            zos.finish();
            zos.close();

            zippedBytes = baos.toByteArray();

            // determine whether the data is too big to be packaged within
            // the CDATA section of the JABBER message
            // It is possible to produce a ZIP file that is bigger than
            // the original file size, if so, do not use ZIPPed data
            if (zippedBytes.length>originalBytes.length)
              fZipit = false;
            else if (zippedBytes.length<=DEFAULT_PAYLOAD_SIZE) {
              fCdata = true;
            }
          }
          // load through list of files
          // file payload
          ePayload  = xmlDoc.createElement(TAG_AGENT_PAYLOAD);

          // set attributes in element
          ePayload.setAttribute( ATTR_FILENAME,
               SystemUtil.extractFileName(srcFile) ); // original filename
          ePayload.setAttribute( ATTR_CONTENT_TRANSFER_ENCODING,
               "base-64" ); // encoding technique defaulted to base-64
          ePayload.setAttribute( ATTR_DESC, sDesc ); // description
          ePayload.setAttribute( ATTR_TIMESTAMP,
                SystemUtil.getDateTime14() ); // time-stamp
          ePayload.setAttribute( ATTR_CHECKSUM,
                "1234567" ); // checksum not implemented yet
          ePayload.setAttribute( ATTR_FILESIZE,
              Long.toString( SystemUtil.fileSize(srcFile) ) ); //file size

          // if performed Gzip, then set "content-type"
          // atribute accordingly
          // store base-64 encoded data in CDATA section
          if (fZipit && flagZip) {
            ePayload.setAttribute(ATTR_CONTENT_TYPE,
                "application/x-zip-compressed");
            if (fCdata)
              ePayload.appendChild(
                xmlDoc.createCDATASection( Base64.encode(zippedBytes) ) );
            else
              ePayload.appendChild( xmlDoc.createCDATASection( "" ) );
          }
          else {
            ePayload.setAttribute(ATTR_CONTENT_TYPE,
              SystemUtil.getContentTypeFromName(srcFile));
            if (fCdata)
              ePayload.appendChild(
              xmlDoc.createCDATASection( Base64.encode(originalBytes) ) );
            else
              ePayload.appendChild( xmlDoc.createCDATASection( "" ) );
          }
          // append list of URLs e.g.
          //     <url>http://server1/GAMMA.bmp</url>
          //     <url>http://server2/GAMMA.bmp</url>
          //     <url>http://server3/GAMMA.bmp</url>
          if ( (arrURL!=null) && (arrURL.length>0)) {
            for (int i=0; i<arrURL.length; i++) {
              if (arrURL[i].length()>0) {
                eURL = xmlDoc.createElement(TAG_URL);
                eURL.appendChild( xmlDoc.createTextNode( arrURL[i]) );
                ePayload.appendChild(eURL);
              }
            }
          }
          // add <AgentPayload> to <AgentJabber>
          eWrapper.appendChild(
                      xmlDoc.createComment( "Payload "+ (++numPayload) ) );
          eWrapper.appendChild(ePayload);
        } catch (IOException e) {
          writeErr( "encodeDataToXML() Error " + e.getMessage() );
          return null;
        }
      }
      else
        return null;
    } // loop through list of <AgentPayload> objects

    // add wrapper to XML document
    xmlDoc.appendChild(eWrapper);
    return xmlDoc;
  }
  else
    return null;
} // encodeDataToXML
```
Figure 131. Procedure to encode binary data to XML

```
//-----------------------------------------------------------------------------
/**
 * read in XML data from a file or a string
 * and search for a particular tag,
 * base-64 decode XML string and save as a file.
 * Note:
 * if destination filename is specified, the filename in the tag attribute
 * is used
 * <AgentJabber filename="hello.bmp">
 *
 * CDATA maybe kept empty if the file size is too big.
 * To retrieve the file from
 * storage location, parse through list of URLs
 * and perform HTTP GET or FTP GET.
 * FTP GET is not implemented yet.
 *
 * @param srcXml source XML file or string
 * @param destFile destination output file, attribute value used if empty
 * @param sTagName tag name to be used
 * @param bFile true if read from file, otherwise it is a string
 * @return list of destination filenames saved to
 * @throws IOException XML exception error
 */
public static ArrayList decodeXMLToData(String srcXml,
                                        String destFile,
                                        String sTagName,
                                        boolean bFile) throws IOException {
  boolean fUnZipit        = false;

  String valFileName      = "";
  String valDesc          = "";
  String valTimeStamp     = "";
  String valContentType   = "";
  String valContentEncode = "";
  long valFileSize        = 0;
  long valCheckSum        = 0;
  String destDir          = "./";

  // determine the destination directory to save the files to
  if ( new File(getDestDir()).isDirectory() ) {
    destDir = getDestDir();
  }

  // list of AgentPayload objects
  ArrayList filesDest = new ArrayList();
  ArrayList lstURL    = new ArrayList();

  if ((srcXml.length()>0) && (sTagName.length()>0)) {
    // read in XML data and create XML document
    org.w3c.dom.Document xmldoc = null;
    if (bFile)
      xmldoc = getXMLDocFromFile(srcXml);
    else
      xmldoc = getXMLDocFromString(srcXml);

    // go to wrapper tag e.g. <AgentJabber>
    NodeList nlWrapper = xmldoc.getElementsByTagName( sTagName );
    for (int idx = 0; idx <nlWrapper.getLength(); idx++) {
      Node child = nlWrapper.item( idx );
      // get list of child nodes under wrapper
      ArrayList cnWrapper =  (ArrayList) getTargetChildNodes(
                              child, new String [] {TAG_AGENT_PAYLOAD} );
      for ( Iterator i=cnWrapper.iterator(); i.hasNext(); ) {
        Node level1   = (Node) i.next();
        String nChild = level1.getNodeName(); // child nodes

        if ( nChild.equalsIgnoreCase( TAG_AGENT_PAYLOAD ) ) {
          Node nPayload = level1;

          try {
            // <AgentJabber filename="hello.bmp">...,
            // variable 'attrFileName' return "hello.bmp"
            if ( nPayload!=null ) {

              // file name
              valFileName   = nPayload.getAttributes().
                              getNamedItem(ATTR_FILENAME).getNodeValue();

              // file description
              valDesc       = nPayload.getAttributes().
                              getNamedItem(ATTR_DESC).getNodeValue();

              // time stamp
              valTimeStamp  = nPayload.getAttributes().
                              getNamedItem(ATTR_TIMESTAMP).getNodeValue();

              // file size
              try {
                valFileSize = Long.parseLong(nPayload.getAttributes().
                              getNamedItem(ATTR_FILESIZE).getNodeValue());
              }
              catch (Exception ex) {
                valFileSize = 0;
              }

              // check sum
              try {
                valCheckSum = Long.parseLong(nPayload.getAttributes().
                              getNamedItem(ATTR_CHECKSUM).getNodeValue());
              }
              catch (Exception ex) {
                valCheckSum = 0;
              }

              // content encoding
              valContentEncode = nPayload.getAttributes().
                 getNamedItem(ATTR_CONTENT_TRANSFER_ENCODING).getNodeValue();

              // content type
              valContentType   = nPayload.getAttributes().
                              getNamedItem(ATTR_CONTENT_TYPE).getNodeValue();
              // determine whether decompression is necessary based on the
              // "content-type" attribute
              if (valContentType.equalsIgnoreCase(
                                             "application/x-zip-compressed"))
                fUnZipit = true;

              // if destination filename is specified,
              // the filename in the tag attribute is used
              destFile = destDir + valFileName;
            }
          }
          catch (Exception ex) {
            ex.printStackTrace();
          }
          //-------------------------------------------------------------------
          // get list of URLs
          lstURL.clear();
          String strURL    = "";
          ArrayList clPayload = (ArrayList)
                    getTargetChildNodes( nPayload, new String [] {TAG_URL} );
          for ( Iterator ii=clPayload.iterator(); ii.hasNext(); ) {
            Node childPayload = (Node) ii.next();
            if ( childPayload.getNodeName().equalsIgnoreCase(TAG_URL) ) {
              try {
                strURL = childPayload.getFirstChild().getNodeValue();
                lstURL.add(strURL);
              }
              catch (Exception ex) { // set to default directory
                strURL = "";
              }
            } // within "url" tag
          } // loop "AgentPayload" children
          //-------------------------------------------------------------------
          // get CDATA value from element, note "<![CDATA[...]]>"
          // are automatically stripped
          // if (nPayload.getFirstChild().getNodeValue()!=null) {
          // something in CDATA
          if (nPayload.getFirstChild().getNodeType()==
              nPayload.CDATA_SECTION_NODE) { // CDATA node?
            StringBuffer binaryData = new
                     StringBuffer( nPayload.getFirstChild().getNodeValue() );

            // perform base64 decoding
            byte[] buffer = Base64.decode(binaryData.toString());

            if (fUnZipit) {
              ByteArrayInputStream bais  = new ByteArrayInputStream(buffer);
              GZIPInputStream zis        = new GZIPInputStream(bais);
              ByteArrayOutputStream baos = new ByteArrayOutputStream();
              int c = -1;
              while ((c = zis.read()) != -1) {
                baos.write(c);
              }
              baos.flush();

              buffer = baos.toByteArray();
            }
            else {
              writeLn("No unzipping required.");
            }

            // write out data to predefined filename provided a filename
            if ( destFile.length()>0) {
              File fDest                = new File(destFile);
              writeLn("writing CDATA to ["+ fDest.getAbsolutePath() +"]");
              BufferedOutputStream bos = new BufferedOutputStream(
                                                new FileOutputStream(fDest));
              bos.write(buffer);
              bos.close();
            }
          }
          else { // no CDATA available, retrieve from URLs
            for (int j=0; j<lstURL.size(); j++) {
              strURL = (String) lstURL.get(j);
              if (strURL.length()>0) {
                //-------------------------------------------------------------
                // HTTP/web server
                if (strURL.toLowerCase().startsWith("http://")) {
                  // e.g. http://www.mango.com/3D.svg
                  String retFile = urlGetFile(strURL, destFile, "");
                  if (retFile.length()>0) {// downloaded file
                    destFile = retFile;
                    break;
                  }
                }
                //-------------------------------------------------------------
                // FTP server
                else if (strURL.toLowerCase().startsWith("ftp://")) {
                  // e.g. ftp://ftp.mango.com/3D.svg
                  // destFile = destDir + "FTP_GET";
                }
                //-------------------------------------------------------------
                // Local/networked file server/location
                else {
                  // e.g. ../../../fruit/3D.svg, \\terra\fruit\3D.svg?


                  File urlFile = new File(strURL);
                  // check that the file can be found
                  if ( urlFile.exists() ) {
                    if ( SystemUtil.filecopy(strURL, destFile) )
                      break;
                  }
                }
                //-------------------------------------------------------------
              }
            } // loop thru' list of URLs
          } // no CDATA available, get from storage server/location
        } // within "AgentPayload"
        String arrURL[] = (String[]) lstURL.toArray( new String[0] );
        AgentPayload agtP = new AgentPayload(destFile, valDesc,
                                            valContentEncode, valContentType,
                                            valTimeStamp,
                                            valFileSize, valCheckSum,
                                            arrURL );
        filesDest.add(agtP);
      } // loop "AgentJabber" children, looking for "AgentPayload"
    } // within "AgentJabber" tag

    return filesDest;
    // for statistical purposes
    /*
    double readBytes       = buffer.length;
    double totalChars      = binaryData.toString().length();
    System.out.println("Encoded " + readBytes + " bytes using " +
                       totalChars + " characters for an average length of " +
                       totalChars/readBytes + " characters.");
    */
  }
  else
    return null;
} // decodeXMLToData
```
Figure 132.   Procedure to decode binary data to XML

APPENDIX F.   GNS.JAVA

The Java class used to convert GEOName Server (GNS) ASCII data files to XML format.

```
//----------------------------------------------------------------------------
/**
 * Filename      : GNS.java
 * Description   : GEOnet Names Server (GNS)
 *                 requires Apache Xerces and util.SystemUtil
 *
 *                 e.g. // convert text-based GNS format to XML form
 *                      GNS.convertTextToXML("C:/test/sn.txt");
 *
 * Created Date  : 29 February 2004
 * Revised Date  : 29 February 2004
 * Course        : Thesis
 * Program       : GNS Object and XML converter
 * Compiler      : JDK 1.4.2 onwards
 * Platform      : Windows 2000/Windows XP
 * @author Lee, Chin Siong Daryl
 * @version 1.0
 */
//----------------------------------------------------------------------------
package main;
import java.io.*;
import java.util.*;

// JAXP packages
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

import org.apache.xml.serialize.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class GNS {
  public static String TAG_GNS     = "GNS";
  public static String ATTR_CTRY   = "ctry";
  public static String ATTR_NUMREC = "numRecords";

  public static String TAG_FEATURE       = "FEATURE";
  public static String ATTR_LAT          = "lat";
  public static String ATTR_LONG         = "long";
  public static String ATTR_DMS_LAT      = "dmsLat";
  public static String ATTR_DMS_LONG     = "dmsLong";
  public static String ATTR_UTM          = "utm";
  public static String ATTR_JOG          = "jog";
  public static String ATTR_GENERIC      = "generic";
  public static String ATTR_SHORT_FORM   = "shortForm";
  public static String ATTR_SORT_NAME    = "sortName";
  public static String ATTR_FULL_NAME    = "fullName";
  public static String ATTR_FULL_NAME_ND = "fullNameND";
  public static String ATTR_MODIFY_DATE  = "modifyDate";
  public static String ATTR_RC           = "rc";
  public static String ATTR_UFI          = "ufi";
  public static String ATTR_UNI          = "uni";
  public static String ATTR_FC           = "fc";
  public static String ATTR_DSG          = "dsg";
  public static String ATTR_PC           = "pc";
  public static String ATTR_ADM1         = "adm1";
  public static String ATTR_ADM2         = "adm2";
  public static String ATTR_CC1          = "cc1";;
  public static String ATTR_CC2          = "cc2";
  public static String ATTR_DIM          = "dim";
  public static String ATTR_NT           = "nt";
  public static String ATTR_LC           = "lc";

  String _sRC;
  String _sUFI;
  String _sUNI;
  String _sLAT;
  String _sLONG;
  String _sDMS_LAT;
  String _sDMS_LONG;
  String _sUTM;
  String _sJOG;
  String _sFC;
  String _sDSG;
  String _sPC;
  String _sCC1;
  String _sADM1;
  String _sADM2;
  String _sDIM;
  String _sCC2;
  String _sNT;
  String _sLC;
  String _sSHORT_FORM;
  String _sGENERIC;
  String _sSORT_NAME;
  String _sFULL_NAME;
  String _sFULL_NAME_ND;
  String _sMODIFY_DATE;

  public GNS(String sRC,
             String sUFI,
             String sUNI,
             String sLAT, String sLONG,
             String sDMS_LAT, String sDMS_LONG,
             String sUTM, String sJOG,
             String sFC, String sDSG,
             String sPC, String sCC1,
             String sADM1, String sADM2,
             String sDIM, String sCC2,
             String sNT, String sLC,
             String sSHORT_FORM,
             String sGENERIC, String sSORT_NAME,
             String sFULL_NAME,
             String sFULL_NAME_ND, String sMODIFY_DATE) {
    _sRC           = sRC;
    _sUFI          = sUFI;
    _sUNI          = sUNI;
    _sLAT          = sLAT;
    _sLONG         = sLONG;
    _sDMS_LAT      = sDMS_LAT;
    _sDMS_LONG     = sDMS_LONG;
    _sUTM          = sUTM;
    _sJOG          = sJOG;
    _sFC           = sFC;
    _sDSG          = sDSG;
    _sPC           = sPC;
    _sCC1          = sCC1;
    _sADM1         = sADM1;
    _sADM2         = sADM2;
    _sDIM          = sDIM;
    _sCC2          = sCC2;
    _sNT           = sNT;
    _sLC           = sLC;
    _sSHORT_FORM   = sSHORT_FORM;
    _sGENERIC      = sGENERIC;
    _sSORT_NAME    = sSORT_NAME;
    _sFULL_NAME    = sFULL_NAME;
    _sFULL_NAME_ND = sFULL_NAME_ND;
    _sMODIFY_DATE  = sMODIFY_DATE;
  } // GNS

  public String getRC() { return _sRC; }     // getRC()
  public String getUFI() { return _sUFI; }   // getUFI()
  public String getUNI() { return _sUNI; }   // getUNI
  public String getLAT() { return _sLAT; }   // getLAT
  public String getLONG() { return _sLONG; } // getLONG
  public String getDMS_LAT() { return _sDMS_LAT; }   // getDMS_LAT
  public String getDMS_LONG() { return _sDMS_LONG; } // getDMS_LONG
  public String getUTM() { return _sUTM; }   // getUTM
  public String getJOG() { return _sJOG; }   // getJOG
  public String getFC() { return _sFC; }     // getFC
  public String getDSG() { return _sDSG; }   // getDSG
  public String getPC() { return _sPC; }     // getPC
  public String getCC1() { return _sCC1; }   // getCC1
  public String getADM1() { return _sADM1; } // getADM1
  public String getADM2() { return _sADM2; } // getADM2
  public String getDIM() { return _sDIM; }   // getDIM
  public String getCC2() { return _sCC2; }   // getCC2
  public String getNT() { return _sNT; }     // getNT
  public String getLC() { return _sLC; }     // getLC
  public String getSHORT_FORM() { return _sSHORT_FORM; } // getSHORT_FORM
  public String getGENERIC() { return _sGENERIC; }       // getGENERIC
  public String getSORT_NAME() { return _sSORT_NAME; }   // getSORT_NAME
  public String getFULL_NAME() { return _sFULL_NAME; }   // getFULL_NAME
  public String getFULL_NAME_ND() { return _sFULL_NAME_ND; } //
getFULL_NAME_ND
  public String getMODIFY_DATE() { return _sMODIFY_DATE; }   //
getMODIFY_DATE

  public String toString() {
    return getLAT() +", "+ getLONG() +", "+ getFULL_NAME()+", "+
getMODIFY_DATE();
```

```
  } // toString
  //------------------------------------------------------------------------
  /**
   * load from text-based GNS data file and save to XML
   * (same filename, different extension)
   * @param srcFile GNS text file
   * @return true=successful, false if failed
   */
  public static boolean convertTextToXML(String srcFile) {
    if (new File(srcFile).exists()) {
      try {
        FileInputStream fis = new FileInputStream(srcFile);
        BufferedReader dis  = new BufferedReader(new InputStreamReader(fis));
        int count           = 0;
        int countError      = 0;
        String sBuf;
        String arrS[];
        ArrayList lst = new ArrayList();

        while ((sBuf = dis.readLine()) != null) {
          arrS = sBuf.split("\t");
          // debugging writeLn( "["+ arrS.length +"]");
          if (count>0) { // skip header
            lst.add( new GNS(arrS[0],  arrS[1],  arrS[2],  arrS[3],  arrS[4],
                             arrS[5],  arrS[6],  arrS[7],  arrS[8],  arrS[9],
                             arrS[10], arrS[11], arrS[12], arrS[13],
arrS[14],
                             arrS[15], arrS[16], arrS[17], arrS[18],
arrS[19],
                             arrS[20], arrS[21], arrS[22], arrS[23], arrS[24]
) );
          }
          count++;
        }
        writeLn("No. of GNS records read from ["+ srcFile +"] is "+ count );
        //------------------------------------------------------------------
        // save as XML
        String fXML = util.SystemUtil.changeFileExt(srcFile, ".xml");
        saveAsXML(lst, fXML);

        writeLn("Generated GNS XML file ["+ fXML +"]" );

        return true;
      } catch(Exception e) {
        writeErr("File error: " + e.getMessage() + " on file " + srcFile);
      }
    }
    return false;
  } // convertTextToXML
  //------------------------------------------------------------------------
  /**
   * create an XML document from list of GNS records
   * @param attrCtry abbreviated country name
   * @param lst list of GNS objects
   * @return XML document
   */
  private static Document createXMLDocument(String attrCtry, ArrayList lst) {
    Element main;
    Element root;
    Element tFeature = null;
    Document _xmlDoc = null;

    try {
      //Create a XML Document
      DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance(); //
DocumentBuilderFactoryImpl.newInstance();
      DocumentBuilder docBuilder = dbFactory.newDocumentBuilder();
      _xmlDoc = docBuilder.newDocument();
    } catch(Exception e) {
      System.out.println("Error " + e);
    }
    // add stylesheet
//    Map PITable = new HashMap(2,(float)0.5); //try this and see what
happens to the output
    //------------------------------------------------------------------------
    root = _xmlDoc.createElement(TAG_GNS);
    root.setAttribute( ATTR_CTRY, attrCtry ); // which country
    root.setAttribute( ATTR_NUMREC, Integer.toString(lst.size()) );
    for (int i=0; i<lst.size(); i++) {
      tFeature = _xmlDoc.createElement(TAG_FEATURE);
      tFeature.setAttribute(ATTR_LAT, ((GNS) lst.get(i)).getLAT() );
      tFeature.setAttribute(ATTR_LONG, ((GNS) lst.get(i)).getLONG() );
      tFeature.setAttribute(ATTR_DMS_LAT, ((GNS) lst.get(i)).getDMS_LAT() );
      tFeature.setAttribute(ATTR_DMS_LONG, ((GNS) lst.get(i)).getDMS_LONG()
);
      tFeature.setAttribute(ATTR_UTM, ((GNS) lst.get(i)).getUTM() );
      tFeature.setAttribute(ATTR_JOG, ((GNS) lst.get(i)).getJOG() );
      tFeature.setAttribute(ATTR_GENERIC, ((GNS) lst.get(i)).getGENERIC() );
      tFeature.setAttribute(ATTR_SHORT_FORM, ((GNS)
lst.get(i)).getSHORT_FORM() );
      tFeature.setAttribute(ATTR_SORT_NAME, ((GNS) lst.get(i)).getSORT_NAME()
);
      tFeature.setAttribute(ATTR_FULL_NAME, ((GNS) lst.get(i)).getFULL_NAME()
);
      tFeature.setAttribute(ATTR_FULL_NAME_ND, ((GNS)
lst.get(i)).getFULL_NAME_ND() );
      tFeature.setAttribute(ATTR_MODIFY_DATE, ((GNS)
lst.get(i)).getMODIFY_DATE() );
      tFeature.setAttribute(ATTR_RC,   ((GNS) lst.get(i)).getRC() );
      tFeature.setAttribute(ATTR_UFI,  ((GNS) lst.get(i)).getUFI() );;
      tFeature.setAttribute(ATTR_UNI,  ((GNS) lst.get(i)).getUNI() );
      tFeature.setAttribute(ATTR_FC ,  ((GNS) lst.get(i)).getFC() );
      tFeature.setAttribute(ATTR_DSG,  ((GNS) lst.get(i)).getDSG() );
      tFeature.setAttribute(ATTR_PC,   ((GNS) lst.get(i)).getPC() );
      tFeature.setAttribute(ATTR_ADM1, ((GNS) lst.get(i)).getADM1() );
      tFeature.setAttribute(ATTR_ADM2, ((GNS) lst.get(i)).getADM2() );
      tFeature.setAttribute(ATTR_CC1,  ((GNS) lst.get(i)).getCC1() );
      tFeature.setAttribute(ATTR_CC2,  ((GNS) lst.get(i)).getCC2() );
      tFeature.setAttribute(ATTR_DIM,  ((GNS) lst.get(i)).getDIM() );
      tFeature.setAttribute(ATTR_NT,   ((GNS) lst.get(i)).getNT() );
      tFeature.setAttribute(ATTR_LC,   ((GNS) lst.get(i)).getLC() );
      root.appendChild(tFeature);
      // debugging writeLn( ((GNS) lst.get(i)).toString() );
    }

    //add to the root Element
    _xmlDoc.appendChild(root);

    return _xmlDoc;
  } // createXMLDocument
  //---------------------------------------------------------------------
  /**
   * save GNS data in XML form
   * @param lstGNS list of GNS objects
   * @param filename file to be saved to
   */
  public static void saveAsXML(ArrayList lstGNS, String filename) {
    try {
      Document doc =
createXMLDocument(util.SystemUtil.extractFileNameOnly(filename),
                                       lstGNS);
      if (doc!=null) {
        OutputFormat outputFormat = new OutputFormat(doc);
        outputFormat.setLineWidth(OutputFormat.Defaults.LineWidth);
        outputFormat.setIndent(OutputFormat.Defaults.Indent);

        XMLSerializer fileSerializer = new XMLSerializer(new
FileWriter(filename), outputFormat);
        fileSerializer.serialize(doc);
      }
      else {
        writeErr("unable to save XML to ["+ filename +"]");
      }
    }
    catch (IOException ioEx) {
      writeErr("Error " + ioEx);
    }
  } // saveScriptXML
  //---------------------------------------------------------------------
  /**
   * write a error messgae to console
   * @param aStr line to be written to console
   */
  public static void writeErr(String aStr) {
    System.err.println(aStr);
  } // writeErr
  //---------------------------------------------------------------------
  /**
   * write a line to console
   * @param aStr line to be written to console
   */
  public static void writeLn(String aStr) {
    System.out.println(aStr);
  } // writeLn
} // GNS
```

LIST OF REFERENCES

[Ant 2004] Apache Ant. http://ant.apache.org/faq.html Accessed on 15 January 2004.

[Ayala 2002] Miguel Arnaldo Ayala, "Execution Level Java Software and Hardware for the NPS Autonomous Underwater Vehicle", Master's Thesis, Naval Postgraduate School, Monterey, California, September 2002. Available at: http://library.nps.navy.mil/uhtbin/cgisirsi/r3TGkbHCIu/99460007/523/3643 Accessed on January 2004.

[Brutzman 1994] Brutzman, D.P., A Virtual World for an Autonomous Underwater Vehicle, PhD Dissertation, Naval Postgraduate School, Monterey, California, December 1994. Available at: http://web.nps.navy.mil/~brutzman/dissertation/ Accessed on January 2004.

[Brutzman 2004] Don Brutzman. X3D Sonar Visualization and Tactical Web Services for Undersea Warfare (USW). Available at http://www.movesinstitute.org/xmsf/projects/sonar-vis/NpsSonarVisualizationTda.ppt. Accessed on February 2004.

[CML] Chemical Markup Language (CML). http://www.xml-cml.org/ Accessed on February 2004.

[DAML] Defense Advanced Research Projects Agency (DARPA) Agent Markup Language (DAML) for agents. http://www.daml.org/ Accessed on February 2004.

[Eclipse 2004] Eclipse Platform. http://www.eclipse.org. Accessed on January 2004.

[Ferber 1999] Ferber, J., Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence. Addison-Wesley, Harlow, England, 1999.

[Girard] Anouck Renee Girard. An Overview of Emerging results in Networked Multi-Vehicle Systems.

[GeoML] Geography Markup Language to describe geographic information. http://www.opengis.org Accessed on February 2004

[Gilles 1998] Gilles Fauconnier, Mark Turner. Conceptual Integration Networks. Available at http://blending.standford.edu. Accessed on March 2004.

[Grunesien 2002] Adrien Gruneisen, Yann Henriet. 3D Model of the Aries Autonomous Underwater Vehicle (AUV), JavaDoc for Dynamics, Software, AUV Mission-Visualization, and AUV Dynamics Control Workbench in Matlab, Naval Postgraduate School, Monterey, California, October 2002.

[Hawkins 2003] Darrin L. Hawkins, Barbara C. Van Leuvan, An XML-based Mission Command Language for Autonomous Underwater Vehicles (AUVs), June 2003. Available at: http://library.nps.navy.mil/uhtbin/cgisirsi/kMbLeal39E/99460007/523/4789

[Hiles 2003] John Hiles. "Cognitive Subjects and Operations: Putting Subjects into Simulations; Moving Agents Out of their Simulation Box". Available at http://www.movesinstitute.org/openhouse2003slides/Hilesopenhouse2003.ppt

[Holden 1995] Holden, Michael J., "ADA Implementation of Concurrent Execution of Multiple Tasks in the Strategic and Tactical Levels of the Rational Behavior Model for the NPS Phoenix Autonomous Underwater Vehicle (AUV)," M.S. thesis, Naval Postgraduate School, Monterey, California 93943, September, 1995. Available at, http://www.cs.nps.navy.mil/research/auv/auv_thesisarchive.html

[Jabber 2004] Jabber. http://www.jabber.org Accessed on September 2003.

[JDK142] Java 2 Platform Standard Edition, v1.4.2 (J2SE). Available at http://java.sun.com/j2se/1.4.2/download.html. Accessed on September 2004.

[JavaCodeConvention 1999] Java Coding Convention. Available at http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html. Accessed on March 2004.

[JEP] Jabber Enhancement Proposals. http://www.jabber.org/jeps/ Accessed on October 2003.

[JEP49] JEP-0049: Private XML Storage. http://www.jabber.org/jeps/jep-0049.html Accessed on January 2004.

[JEP71] JEP-0071: XHTML-IM. http://www.jabber.org/jeps/jep-0049.html Accessed on January 2004.

[JiveSoftware 2003] Jive Software open-source XMPP client library for instant messaging and presence. http://www.jivesoftware.com/xmpp/smack/. Accessed on September 2003.

[JMS] Sun Java Message Service. http://java.sun.com/products/jms/. Accessed on October 2003.

[Mahmoud 2002] Qusay H. Mahmoud. Compressing and Decompressing Data using Java. Accessed on February 2002.

[MathML] MathML for mathematics. http://www.w3.org/Math/ Accessed on February 2004.

[Mozilla 2004] Mozilla Project. http://www.mozilla.org/ Accessed on February 2004.

[MsMQ] Microsoft Message Queuing. www.microsoft.com/msmq/default.htm Accessed on October 2003.

[Netbeans 2004] NetBeans Platform. http://www.netbeans.org. Accessed on January 2004.

[Oceans 2000] David B. Marco, Anthony J. Healey. Current Developments in Underwater Vehicle Control and Navigation: The NPS ARIES AUV, 2000. Available at http://web.nps.navy.mil/~me/healey/papers/Oceans2000.pdf

[OSI 2004] Open Source Initiative, Non-Profit Corporation, 2002, "Definition and Rationale", http://www.opensource.org (Accessed February 2004).

[Pentakalos 2001] Odysseas Pentakalos. Java Tip 117: Transfer binary data in an XML document. http://www.javaworld.com/javaworld/javatips/jw-javatip117.html (Accessed on January 2004).

[Polycarpou 2001] Marios M. Polycarpou. Ohio State University. Cooperative Control of Distributed Multi-Agent Systems.

[RFC 821] RFC 821 - Simple Mail Transfer Protocol. http://www.faqs.org/rfcs/rfc821.html (Accessed on December 2003).

[RFC 959] RFC959 - File Transfer Protocol. http://www.w3.org/Protocols/rfc959/Overview.html or http://www.faqs.org/rfcs/rfc959.html (Accessed on December 2003).

[RFC 1867] RFC 1867 - Form-based File Upload in HTML. http://www.faqs.org/rfcs/rfc1867.html (Accessed on December 2003)

[RFC 2045] RFC 2045 (Base64 Encoding). http://www.ietf.org/rfc/rfc2045.txt (Accessed on December 2003).

[RFC 2660] RFC 2660 (Secure HyperText Transfer Protocol) http://www.ietf.org/rfc/rfc2660.txt. (Accessed on January 2004).

[RFC 2779] RFC2779 - Instant Messaging / Presence Protocol Requirements. http://www.jabber.org/ietf/ (Accessed on January 2004).

[SFTP 2002] Secure FTP 101. http://www.intranetjournal.com/articles/200208/se_08_14_02a.html (Accessed on January 2004).

[Reimers 1995] Reimers, S. "Towards Internet Protocol Over Seawater: Forward Error Correction Using Hamming Codes for Reliable Acoustic Telemetry", MS Thesis, Naval

Postgraduate School, Monterey, California. September 1995.

[Rhymbox 2004] RhymBox Jabber Client - Instant Messaging For XMPP/Jabber. http://www.rhymbox.com/. Accessed on 15 January 2004.

[Schema 2004] XML Schema. http://www.w3.org/XML/Schema Accessed on February 2004.

[SensorML] Sensor Markup Language (SensorML) for sensors. http://vast.uah.edu/SensorML/ Accessed on February 2004.

[Serin 2003] Serin, E., "Design and Test of the Cross-Format Schema Protocol (XFSP) for Networked Virtual Environment", Master's Thesis, Naval Postgraduate School, Monterey, California, March 2003.

[Shankar 2002] Gowri Shankar. Embed binary data in XML documents three ways. http://www-106.ibm.com/developerworks/xml/library/x-binary/?open&l=136,t=gr,p=xb2b Accessed on February 2002.

[SVG 2004] Scalable Vector Graphics (SVG). http://www.w3.org/Graphics/SVG/. Accessed on January 2004.

[Turner] Roy M. Turner. University of New Hampshire. Handling Unanticipated Events in Single and Multiple AUV Systems.

[Turner 2002] Gilles Fauconnier, Mark Turner. The Way We Think: Conceptual Blending and The Mind's Hidden Complexities.

[Websphere] IBM Websphere MQ. http://www-306.ibm.com/software/integration/wmq/

[Wheless] Glen H. Wheless. Old Dominion University. The Use of Collaborative Virtual Environments in the Mine Countermeasures Mission.

[XMPP 2004] Extensible Messaging and Presence Protocol (XMPP). http://xml.coverpages.org/xmpp.html. Accessed on December 2003.

[XSL 2004] Extensible Stylesheet Language. http://www.w3.org/TR/NOTE-XSL.htm. Accessed on February 2004.

[XML 1999] XML in 10 points, http://www.w3.org/XML/1999/XML-in-10-points Accessed on January 2004.

[XML 2004] Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/2004/REC-xml-20040204 Accessed on February 2004.

[XSLT 2004] Extensible Stylesheet Language Transformation. http://www.w3.org/TR/xslt. Accessed on February 2004.

[Xalan 2004] Apache Xalan. http://xml.apache.org. Accessed on February 2004.

[X3D] Extensible 3D (X3D) Graphics. http://www.web3d.org/x3d.html Accessed on February 2004.

[X3DHints 2004] X3D Scene Authoring Hints. Available at http://www.web3d.org/TaskGroups/x3d/translation/examples/X3dSceneAuthoringHints.html#NamingConventions. Accessed on September 2003.

[XJ3D 2004] The Xj3D Project. http://www.xj3d.org Accessed on February 2004.

[XTC 2004] Don Brutzman, Don McGregor, Daniel A. DeVos and Chin Siong Lee. XML-based Tactical Chat (XTC): Requirements, Capabilities and Preliminary Progress, January 2004. Available at http://www.movesinstitute.org/xmsf/projects/XTC/XmlTacticalChat2004January28.pdf

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST